Applicant: Thomas R. Firman
Filed: February 14, 2001
For: AUTOMATIC ASSEMBLY OF VOICE CONTROL INFORMATION
Attorney of Record: David L. Feigenbaum, Reg. No. 30,378
                    Fish & Richardson P.C.
                    225 Franklin Street
                    Boston, MA 02110

Appendix C

```
/*
** File: CONTEXT.C
**
** This module determines the correct context for spoken utterances to
** be executed in.
**
** Public functions:   ContextNewLang
**                     ContextCheck
**                     ContextListAdd
**                     ContextListSelect
**
** Exported functions: ContextEnumProc
**
** Private functions:  ContextListInit
**                     ContextAdd
**                     HasKey
**                     ContextAddAccel
**                     ContextAddMenu
**                     GetActiveLang
**                     AddLang
**                     AddLngCommands
**                     AddScrollBarCommands
**                     ContextAddScrollBars
**                     ContextAddWindSysCom
**                     ContextAddPMGroup
**                     ContextAddWind
**                     ContextAddPopupMenu
**                     StringGetSysChar
**                     ContextPakWind
**                     ContextPakMenu
**                     ContextPakSysCom
**                     ContextPakScroll
**                     ContextPakWindDebug
**                     ContextPakDebug
**                     ContextPak
**
*********************************************************************************
********************************************************************************/


#define WIN31            // need this to use extended 3.1 functionality
#include <windows.h>

#include <toolhelp.h>

#include "vtools.h"
#include "vc.h"

#ifdef DEBUG_DLG
int ContextTabs[3];
#endif

/*-------------------------------------------------------------------
|
| How is one window related to another.
|
```

```
*/
enum
{
        CW_HASFOCUS   = 1,
        CW_PARENTLEVEL = 2,
};


/*------------------------------------------------------------------
|
| Description for item in  context list.
|
*/
typedef struct tagCONTEXTITEM
{

        enum
        {          // What type of context info is it.
                CON_WIND,          //   It is a window or a control.
                CON_ICON,          //   An iconized window.
                CON_SYSCOM,          //   It is a universal window control. min/max/sys
                CON_SCROLL,          //   Scrolling commands.
                CON_MENUPOPUP,          //   A menu bar item that will popup.
                CON_MENU,          //   A menu item in the active menu.
                CON_ACCEL,          //   A short cut key.
                CON_LAUNCH,          //   Executable item.
                CON_MACRO,
        } conType;

        int  iLevel;          // The window group/probability level.
        HWND hwnd;          // Handle to the window associated.

        union
        {
                struct
                {
                        enum
                        {          // Is it a class we know about.
                                CWC_STATIC,
                                CWC_BUTTON,
                                CWC_LISTBOX,
                                CWC_COMBOBOX,
                                CWC_EDIT,
                                CWC_SCROLLBAR,
                                CWC_PMGROUP,
                                CWC_MDICLIENT,
                                CWC_CHILD,          // Other child
                                CWC_GROUPBOX,          // Special case
                                CWC_POPUP,          // Other popup
                        } cwc;
                        BOOL  bForList;
                        LPSTR szName;
                } Window;

                int SysCom;          // System command id.

                int ScrlCom;          // Scroll Interfase
```

```c
· struct
{
        HMENU hMenu;
        int   iEntry;
        int   iKeyPos;          // How far down is it not counting separators
} MenuPop;

struct
{
        HMENU hMenu;                            // Handle of the menu
        WORD id;                                // Item ID
        LPSTR szName;           // Alias name from Lang
} Menu;

struct
{
        HMENU hMenu;
        WORD  id;                               // Item ID
} Acc;

struct
{
        PSTR szTitle;           // Title
        PSTR szFile;            // Command string
} PMItem;               // PMItem string for CON_LAUNCH


        LPMACRO pMacro;         // Macro from language

} u;

struct tagCONTEXTITEM * pciNext;        // Next item in the list.

} CONTEXTITEM;

/*--------------------------------------------------------------
|
| Scroll bar types.
|
*/
#define SCRLS_HORZ (0x8000)
#define SCRLS_WIN  (0x4000)
#define SCRLS_MDI  (0x2000)
#define SCRLS_ACT  (~(SCRLS_HORZ | SCRLS_WIN | SCRLS_MDI))

/*--------------------------------------------------------------
|
| Scroll present mask
|
*/
#define SCRLM_HORZ  (0x0001)            // Is horz scroll present
#define SCRLM_VERT  (0x0002)            // Is vert scroll present
#define SCRLM_HMDI  (0x0004)            // Is MDI Workspace scroll present
#define SCRLM_VMDI  (0x0008)
```

```
/*-----------------------------------------------------------------
|
| Context List.
|
*/
_LOCAL CONTEXTITEM * pciFirst = NULL;
_LOCAL CONTEXTITEM * pciLast = NULL;

_LOCAL unsigned iCheckSum = (UINT)-1;        // Keep a check sum of the context.

_LOCAL HWND  hwndFocus = NULL;                       // Focus window
_LOCAL HWND  hwndActive = NULL;                      // Active window
_LOCAL HWND  hwndParent;                             // Current parent interogated.
_LOCAL HWND  hwndPrvParent;                          // This was the previous parent.

_LOCAL int   iCaptionLen;                // The longest context caption length.
_LOCAL int   iDebugCapLen;
_LOCAL int   iGroupLevel;           // The context group number.

_LOCAL FARPROC lpprocContext = NULL;

_LOCAL char  szCaptionBuf[2 * MAX_SYMBOL_LENGTH + 50];      // Caption buffer.

_LOCAL LPLANG pLangCur = NULL;

/*-----------------------------------------------------------------
|
| These are switches
|
*/
_LOCAL BOOL  bChildSysMenu;          // Child sys commands used ?
_LOCAL HWND  hwndMenuSysPop;          // Is the sys menu popped up ?
_LOCAL BOOL  bMenuBarExist;
_LOCAL BOOL  bMenuPopExist;          // Is there a popup menu active.
_LOCAL int   iScrollMask;          // Is scroll present  mask.

/*-----------------------------------------------------------------
|
| These are predefinned classes.
|
*/
_LOCAL PSTR szPredefClass[] =
{
        "Static",
        "Button",
        "ListBox",
        "ComboBox",
        "Edit",
        "ScrollBar",
        "PMGroup",                    // Program manager groups.
        "MDIClient",
};

/*-----------------------------------------------------------------
|
| FUNCTION    _LOCAL void ContextListInit(void)
```

```
|
| DESCRIPTION Clear the previous context list.
|
| PARAMETERS  None.
|
| RETURN     None.
|
*/
_LOCAL void ContextListInit(void)
{
        /* Delete old context list
        */
        while (pciFirst != NULL)
        {
                pciLast = pciFirst->pciNext;
                if (pciFirst->conType == CON_LAUNCH)
                {
                        /* We allocate these string
                        */
                        StringNearDestroy(pciFirst->u.PMItem.szTitle);
                        StringNearDestroy(pciFirst->u.PMItem.szFile);
                }
                Nfree(pciFirst);
                pciFirst = pciLast;
        }

        /* Reset the checking environment.
        */
        iCheckSum = 0;

        /* Leave 0 for the lang overrides.
        */
        iGroupLevel = 1;

        /* A pop up menu is on top.
        */
        hwndMenuSysPop  = NULL;

        /* The menu bar has been read ?
        */
        bMenuBarExist    = FALSE;

        /* Child sys commands used ?
        */
        bChildSysMenu = FALSE;

        /* No scroll commands yet
        */
        iScrollMask     = 0;
}

/*-------------------------------------------------------------------
|
| FUNCTION   _LOCAL BOOL ContextAdd(hwnd, conType)
|
| DESCRIPTION Add an item of context info to the list.
```

```
|         Filling in the union feilds is up to the caller.
|
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|          int conType -
|
| RETURN     TRUE if success
|
*/
_LOCAL BOOL ContextAdd(HWND hwnd, int conType)
{
        CONTEXTITEM * pci;
        int c;

        if (pciLast != NULL)
        {
                /* Checksum the previous.
                */
                for (c = 0; c < sizeof(CONTEXTITEM); c ++)
                {
                        iCheckSum += ((PSTR) pciLast)[c];
                }
        }

        /* Must have a window ?
        */
        if (hwnd == NULL)
                return(FALSE);

        /* Allocate struct
        */
        pci = (CONTEXTITEM*) Nmalloc(sizeof(CONTEXTITEM));
        if (pci == NULL)
                return(FALSE);

        /* Set basic vars
        */
        pci->conType = conType;
        pci->iLevel = iGroupLevel;
        pci->hwnd = hwnd;

        /* Insert it after the pciLast.
        */
        if (pciFirst == NULL || pciLast == NULL)
        {
                /* At the start.
                */
                pci->pciNext = pciFirst;

                /* save top.
                */
                pciFirst = pci;
        }
        else
        {
                /* Insert after pciLast.
                */
```

```
                         pci->pciNext = pciLast->pciNext;

                         /* Add to end.
                         */
                         pciLast->pciNext = pci;
                 }

                 /* The current pointer.
                 */
                 pciLast = pci;

                 /* Return true so we continue enumerating.
                 */
                 return(TRUE);
        }


/*----------------------------------------------------------------------
|
| FUNCTION    _LOCAL BOOL HasKey(hMenu, iPos)
|
| DESCRIPTION Check if the given menu has accelerator key.
|                         We check only \t, \a, or \b presents in the string
|
|
| PARAMETERS  HWND hMenu - Specifies handle to the given menu.
|         int  iPos  - specifies posititon in the menu
|
| RETURN
|
*/
_LOCAL BOOL HasKey(HMENU hMenu, int iPos)
{
        int i;

        if(! GetMenuString(hMenu, iPos, szCaptionBuf, sizeof(szCaptionBuf) - 1,
MF_BYPOSITION))
        {
                /* No text at all
                */
                return(FALSE);
        }

        for (i = 0; i < lstrlen(szCaptionBuf) - 1; i ++)
        {
                if (szCaptionBuf[i] == '\t' ||   // For Windows Apps
                        szCaptionBuf[i] == '\a' ||
                        szCaptionBuf[i] == '\b')     // For Microsoft Apps
                {
                        /* Has TAB or ...
                        */
                        return(TRUE);
                }
        }
        return(FALSE);

}
```

```
/*-----------------------------------------------------------------
|
| FUNCTION    _LOCAL void ContextAddAccel(HWND hwnd, HMENU hMenu)
|
| DESCRIPTION Add the menu options to the context list.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|             HWND hMenu - Specifies handle to the given menu.
|
| RETURN      None.
|
*/
_LOCAL void ContextAddAccel(HWND hwnd, HMENU hMenu)
{
        int  iPos;
        int  items;
        WORD State;

        if (hMenu == NULL)
        {
                /* No menu
                */
                return;
        }

        /* For all items
        */
        items = GetMenuItemCount(hMenu);
        for (iPos = 0; iPos < items; iPos ++)
        {
                State = GetMenuState(hMenu, iPos, MF_BYPOSITION);
                if (State == -1)
                        break;
                if (State & MF_POPUP )
                {
                        /* Check submenu
                        */
                        ContextAddAccel(hwnd, GetSubMenu(hMenu, iPos));
                }
                else if (!(State & (MF_DISABLED | MF_GRAYED | MF_BITMAP |
MF_OWNERDRAW)))
                {
                        if (HasKey(hMenu, iPos))
                        {
                                /* Add accelerator now
                                */
                                if (! ContextAdd(hwnd, CON_ACCEL))
                                        return;
                                pciLast->u.Acc.hMenu = hMenu;

                                /* We use position as an ID
                                */
                                pciLast->u.Acc.id = GetMenuItemID(hMenu, iPos);
                        }
                }
```

```
            }

    }

/*----------------------------------------------------------------------
|
| FUNCTION    _LOCAL BOOL ContextAddMenu(HWND hwnd, HMENU hMenu)
|
| DESCRIPTION Add the menu options to the context list.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|         HWND hMenu - Specifies handle to the given menu.
|
| RETURN      None.
|
*/
_LOCAL BOOL ContextAddMenu(HWND hwnd, HMENU hMenu)
{
        int i;
        int items;
        WORD State;
        int numseparators;

        if (hMenu == NULL)
        {
                /* No menu
                */
                return(FALSE);
        }

        /* For all items
        */
        items = GetMenuItemCount(hMenu);
        numseparators = 0;
        for (i = 0; i < items; i ++)
        {
                State = GetMenuState(hMenu, i, MF_BYPOSITION);
                if (State == -1)
                        break;
                if (! ContextAdd(hwnd, (State & MF_POPUP) ? CON_MENUPOPUP :
CON_MENU))
                        return(FALSE);

                /* Popups return different values
                */
                if (!(State & MF_POPUP))
                {
                        /* Skip separator
                        */
                        if (State & MF_SEPARATOR)
                                numseparators++;
                }

                if (pciLast->conType == CON_MENUPOPUP)
                {
                        /* Store the entry number.
```

```c
                            */
                            pciLast->u.MenuPop.iEntry = i;
                            pciLast->u.MenuPop.iKeyPos = i - numseparators;
                            pciLast->u.MenuPop.hMenu = hMenu;
                    }
                    else
                    {
                            /* Store ID.
                            */
                            pciLast->u.Menu.id = GetMenuItemID(hMenu, i);
                            pciLast->u.Menu.hMenu = hMenu;

                    }
            }
            return(TRUE);
}


/*----------------------------------------------------------------------
|
| FUNCTION    void ContextNewLang(pLangEdit)
|
| DESCRIPTION Change macro language.
|
| PARAMETERS  LPLANG pLangEdit - Specifies pointer to the new language.
|
| RETURN      None.
|
*/
void ContextNewLang(LPLANG pLangEdit)
{
        char szFile[MAXFILENAME + 1];

        /* Destroy old language if present
        */
        LangChainDestroy(pLangCur);

        if (pLangEdit == NULL)
        {
                /* Try to open users language
                */
                IniGetUserFile(szFile);
                lstrcat(szFile, ".LNG");
                pLangCur = LangLoad(szFile);
        }
        else
        {
                /* Try to copy from editor
                */
                pLangCur = LangChainMake(pLangEdit);

        }
        if (pLangCur == NULL)
        {
                /* Try to open default language
                */
                IniGetLangFile(szFile);
```

```
                    pLangCur = LangLoad(szFile);
        }

}

/*-----------------------------------------------------------------------
|
| FUNCTION    _LOCAL LPLANG GetActiveLang()
|
| DESCRIPTION A new task has been loaded so load new language
|         or the default langauge.
|
| PARAMETERS  None.
|
| RETURN     Pointer to the app specific language.
|
*/
_LOCAL LPLANG GetActiveLang()
{

        static HWND   hwndPrevActive = NULL;
        static LPLANG pActiveLang = NULL;
        HANDLE hTask;
        TASKENTRY te;
        char szFile[MAXFILENAME + 1];

        /* Active window changed
        */
        if (hwndPrevActive != hwndActive)
        {
                /* Save currently active window for the next call
                */
                hwndPrevActive = hwndActive;

                /* Get task handle
                */
                hTask = GetWindowTask(hwndActive);

                if (hTask == NULL)
                {
                        /* No task ?!
                        */
                        pActiveLang = NULL;
                }
                else
                {
                        /* Get module name
                        */
                        te.dwSize = (DWORD) sizeof(te);
                        TaskFindHandle((TASKENTRY FAR *) &te, hTask);
                        GetModuleFileName(te.hModule, (LPSTR)szFile, sizeof(szFile) - 1);

                        /* Try to find language
                        */
                        for (pActiveLang = pLangCur->pNext;   pActiveLang != NULL;
pActiveLang = pActiveLang->pNext)
```

```c
                    {
                            if (! lstrcmpi(szFile, pActiveLang->szFile))
                            {
                                    /* Here it is
                                    */
                                    break;
                            }
                    }
            }
    }

    /* Return pointer to the language or NULL
    */
    return(pActiveLang);

}
```

```
/*-------------------------------------------------------------------
|
| FUNCTION    _LOCAL void AddLang(pLang, hwnd, szClass, szWndText, bMenuPopExist )
|
| DESCRIPTION Add macro command from the language
|
| PARAMETERS  LPLANG pLang  - Specifies pointer to the language.
|         HWND hwnd  - Specifies handle to the window we are looking at.
|         PSTR szClass   - Specifies pointer to the class name string.
|         PSTR szWndText - Specifies pointer to the windows title.
|         BOOL bMenuPopExist - TRUE if popup menu on the screen.
|
| RETURN     None.
|
*/
```

```c
_LOCAL void AddLang(LPLANG pLang, HWND hwnd, PSTR szClass, PSTR szWndText, BOOL
bMenuPopExist )
{
        LPGROUP pGroup;
        LPMACRO pMacro;
        HWND hwndMacro;

        if (pLang == NULL)
                /* No language selected
                */
                return;

        /* Try to find proper group
        */
        for (pGroup = pLang->pGroup; pGroup != NULL; pGroup = pGroup->pNext)
        {
                if (
                        /* Default group
                        */
                        (pGroup->szClass == NULL && szClass == NULL)
                        /* Class group
                        */
                        ||(( pGroup->szClass != NULL && szClass != NULL && !
lstrcmp(pGroup->szClass, szClass)
```

```c
                                ) && (pGroup->szWndText == NULL || ! lstrcmp(pGroup->szWndText,
szWndText))))
                {
                        /* Work with macros if the group has been found
                        */
                        for (pMacro = pGroup->pMacro; pMacro != NULL; pMacro = pMacro-
>pNext)
                        {
                                hwndMacro = hwnd;
                                switch (pMacro->cmdType)
                                {
                                        case CMD_WNDNAME:
                                        {
                                                /* Set allias name for the window
                                                */
                                                CONTEXTITEM * pci;
                                                char szBuf[MAXSTRING + 1];

                                                /* Look through the whole list
                                                */
                                                for (pci = pciFirst; pci != NULL; pci = pci-
>pciNext)

                                                {
                                                        /* We need CON_WIND or CON_ICON

                                                        */
                                                        if (pci->conType == CON_WIND
|| pci->conType == CON_ICON)
                                                        {
                                                                GetClassName(pci->hwnd,
szBuf, sizeof(szBuf)-1);

                                                                /* Compare class name and

                                                                */
child ID
                                                                if (pMacro->itemid ==
GetWindowWord(pci->hwnd, GWW_ID) &&
                                                                        ! lstrcmp(szBuf,
pMacro->szWndClass))
                                                                {
                                                                        /* Window shouldn't
have allias yet
                                                                        */
                                                                        if (pci-
>u.Window.szName == NULL)
                                                                        {
                                                                                /* Set it
                                                                                */
                                                                                pci-
>u.Window.szName = pMacro->szName;
                                                                                break;
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                        break;
```

```
                                                                        }

                                                            case CMD_MENUNAME:
                                                            {
                                                                    /* Set allias name for the menu item
                                                                    */
                                                                    CONTEXTITEM * pci;

                                                                    for (pci = pciFirst; pci != NULL; pci = pci-
         >pciNext)

                                                                    {
                                                                            /* We need CON_MENU with the same

                                                                            */
                                                                            if (pci->conType == CON_MENU
         ID

                                                                                    pMacro->itemid == pci-
         &&
         >u.Menu.id)
                                                                            {
                                                                                    /* Item shouldn't have allias yet
                                                                                    */
                                                                                    if (pci->u.Menu.szName ==
         NULL)
                                                                                    {
                                                                                            /* Set it
                                                                                            */
                                                                                            pci->u.Menu.szName =
         pMacro->szName;
                                                                                            break;
                                                                                    }
                                                                                    break;
                                                                            }
                                                                    }
                                                                    break;
                                                            }

                                                            case CMD_MOUSE :
                                                            case CMD_JOURNAL :
                                                            {
                                                                    /* For mouse and journal macro we need to find

                                                                    */
                                                                    CONTEXTITEM * pci;
                                                                    char szBuf[MAXSTRING + 1];

                                                                    /* Class name of the window is the main
         descriptor
                                                                    */
                                                                    if (pMacro->szWndClass)
                                                                    {
         window to play to it
                                                                            hwndMacro = NULL;

                                                                            /* Look through the whole list
                                                                            */
```

```
pci->pciNext)

CON_WIND)

and child ID

>hwnd, szBuf, sizeof(szBuf)-1);

GetWindowWord(pci->hwnd, GWW_ID) &&

pMacro->szWndClass))

found it

pci->hwnd;
```

menu on the screen

```
                                    }

                                }

                            }

                        }

                    }
```

```
for (pci = pciFirst; pci != NULL; pci =

{
        if (pci->conType ==

        {
                /* Compare class name

                */
                GetClassName(pci-

                if (pMacro->itemid ==

                        ! lstrcmp(szBuf,

                {
                        /* we have

                        */
                        hwndMacro =

                        break;
                }
        }
}
if (hwndMacro == NULL)
{
        /* No window
        */
        break;
}
```

```
default:
        if (bMenuPopExist)
                /* Can not do anything while popup

                */
                break;
        if (! ContextAdd(hwndMacro, CON_MACRO))
                /* Not enough memory
                */
                return;
        /* Add it
        */
        pciLast->u.pMacro = pMacro;
```

```
/*-------------------------------------------------------------------------
|
| FUNCTION    _LOCAL void AddLngCommands(hwnd, szClass, szWndText, bMenuPopExist )
|
| DESCRIPTION Add macro command.
|
| PARAMETERS  HWND hwnd  - Specifies handle to the window we are looking at.
|         PSTR szClass   - Specifies pointer to the class name string.
|         PSTR szWndText - Specifies pointer to the windows title.
|         BOOL bMenuPopExist - TRUE if popup menu on the screen.
|
| RETURN     None.
|
*/
_LOCAL void AddLngCommands(HWND hwnd, PSTR szClass, PSTR szWndText, BOOL
bMenuPopExist )
{

        if (pLangCur == NULL)
        {
                /* No language at all
                */
                return;
        }

        /* Application specific    language
        */
        AddLang(GetActiveLang(), hwnd, szClass, szWndText, bMenuPopExist);

        /* Global language
        */
        AddLang(pLangCur, hwnd, szClass, szWndText, bMenuPopExist);
}


/*-------------------------------------------------------------------------
|
| FUNCTION    _LOCAL void AddScrollBarCommands(hwnd, ScrollMask, iCheckMask)
|
| DESCRIPTION Create scroll bar command.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|         int  ScrollMask - Specifies scroll mask.
|         int  iCheckMask - Specifies check mask.
|
| RETURN     None.
|
*/
_LOCAL void AddScrollBarCommands(HWND hwnd, int ScrollMask, int iCheckMask)
{

        /* Scroll command with this name shouldn't be in the list twice
        */
        if (! (iScrollMask & iCheckMask))
        {
                /* This is first one
```

```
                    */
          iScrollMask |= iCheckMask;

          if (! ContextAdd(hwnd, CON_SCROLL))
                  /* Not enough memory
                   */
                  return;
          pciLast->u.ScrlCom = SB_LINEUP | ScrollMask;

          if (! ContextAdd(hwnd, CON_SCROLL))
                  /* Not enough memory
                   */
                  return;
          pciLast->u.ScrlCom = SB_LINEDOWN | ScrollMask;

          if (! ContextAdd(hwnd, CON_SCROLL))
                  /* Not enough memory
                   */
                  return;
          pciLast->u.ScrlCom = SB_PAGEUP | ScrollMask;

          if (! ContextAdd(hwnd, CON_SCROLL))
                  /* Not enough memory
                   */
                  return;
          pciLast->u.ScrlCom = SB_PAGEDOWN | ScrollMask;

      }

}

/*---------------------------------------------------------------------
|
| FUNCTION    _LOCAL void ContextAddScrollBars(hwnd, Style, cwc)
|
| DESCRIPTION Add scroll bar commands.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|       LONG Style - Specifies windows style
|       int cwc - Specifies window type.
|
| RETURN      None.
|
*/
_LOCAL void ContextAddScrollBars(HWND hwnd, LONG Style, int cwc)
{
        switch (cwc)
        {
                case CWC_MDICLIENT:
                        if (Style & WS_VSCROLL)
                        {
                                AddScrollBarCommands(hwnd, SCRLS_MDI, SCRLM_VMDI);
                        }
                        if (Style & WS_HSCROLL)
                        {
```

```
                                AddScrollBarCommands(hwnd, SCRLS_MDI | SCRLS_HORZ,
SCRLM_HMDI);
                        }
                        break;

                case CWC_SCROLLBAR :
                        if (Style & SBS_VERT)
                        {
                                AddScrollBarCommands(hwnd, SCRLS_WIN, SCRLM_VERT);
                        }
                        else
                        {
                                AddScrollBarCommands(hwnd, SCRLS_WIN | SCRLS_HORZ,
SCRLM_HORZ);
                        }
                        break;

                default:
                        if (Style & WS_VSCROLL)
                        {
                                AddScrollBarCommands(hwnd, 0, SCRLM_VERT);
                        }
                        if (Style & WS_HSCROLL)
                        {
                                AddScrollBarCommands(hwnd, SCRLS_HORZ,
SCRLM_HORZ);
                        }
        }
}

/*------------------------------------------------------------------------
|
| FUNCTION   _LOCAL void ContextAddWindSysCom(hwnd, Style)
|
| DESCRIPTION Add system type commands for the window.
|
| PARAMETERS  HWND hwnd  - Specifies handle to the given window.
|        LONG Style - Specifies windows style
|
| RETURN     None.
|
| NOTE       Maximized MDI children are strange.
|        The sys menu/restore is in the main menu of parent.
|        They will not register normal WS_SYSMENU and restore boxes.
|        Microsoft Excel violates even these rules !
|        It will not set the WS_MAXIMIZE bit !
|
*/
_LOCAL void ContextAddWindSysCom(HWND hwnd, LONG Style)
{
        if (! (Style & WS_CHILD) || ! (Style & WS_MAXIMIZE))
        {
                /* Does the window have system command menu ?
                */
                if (! (Style & WS_SYSMENU))
                        return;
```

```c
}
else
{
        /* Can we get one ?
        */
        if (GetSystemMenu(hwnd, FALSE) == NULL)
                return;
}

/* Already got sysmenu type stuff ?
*/
if (bChildSysMenu && (Style & WS_CHILD))
        return;
bChildSysMenu = TRUE;

/*  Check to see if sys menu is already popped up.
*/
if (hwndMenuSysPop == hwnd)
        /* Already popped.
        */
        return;

/* Option to pull down the sys menu.
*/
if (! ContextAdd(hwnd, CON_SYSCOM))
        return;
/* The menu itself.
*/
pciLast->u.SysCom = SC_KEYMENU;

/* If the window is iconic then the others are not really available.
** Although they will say they are.
*/
if (Style & WS_ICONIC)
        return;

/* Option to close the window or app
** This is equiv. to double click on sys menu box.
*/
if (! ContextAdd(hwnd, CON_SYSCOM))
        return;
pciLast->u.SysCom = SC_CLOSE;

/* Get the min/max controls seperatly for now.
*/
if (Style & WS_MINIMIZEBOX)
{
        if (! ContextAdd(hwnd, CON_SYSCOM))
                return;
        pciLast->u.SysCom = SC_MINIMIZE ;
}

/* If the window is maximzed then we need a restore box.
*/
if (Style & WS_MAXIMIZEBOX)
{
```

```
               if (! ContextAdd(hwnd, CON_SYSCOM))
                        return;
               pciLast->u.SysCom = (Style & WS_MAXIMIZE) ? SC_RESTORE :
SC_MAXIMIZE ;
        }
}


/*---------------------------------------------------------------------
|
| FUNCTION    _LOCAL void ContextAddPMGroup(hwnd, Style)
|
| DESCRIPTION Add content of Program Manager Group
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|        LONG Style - Specifies windows style
|
| RETURN     None.
|
*/
_LOCAL void ContextAddPMGroup(HWND hwnd, LONG Style)
{
        SHELLITEM si;
        BOOL  bRet;

        if (Style & WS_ICONIC)
        {
                /* We dont look inside iconic window, user cannot either
                */
                return ;
        }
        /* Window text is a group name
        */
        GetWindowText(hwnd, szCaptionBuf, sizeof(szCaptionBuf) - 1);

        /* Enumerate PM items inside the group
        */
        bRet = ShellGetFirstItem(&VCTalk, szCaptionBuf, &si);
        while (bRet)
        {
                /* We need command string to execute
                */
                if (si.szFile)
                {
                        if (! ContextAdd(hwnd, CON_LAUNCH))
                                /* not enough memory
                                */
                                return;

                        /* Title is the name, file is the command string
                        */
                        pciLast->u.PMItem.szTitle = StringNearMake(si.szTitle);
                        pciLast->u.PMItem.szFile = StringNearMake(si.szFile);
                }

                /* Next one ?
                */
```

. bRet = ShellGetNextItem(&VCTalk, &si);

   }

}

```
/*------------------------------------------------------------------------
|
| FUNCTION    _LOCAL BOOL ContextAddWind(hwnd, checktype)
|
| DESCRIPTION Check the window for useful context info.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|          int  checktype - what type are we looking at. CW_*
|
| RETURN      TRUE if success.
|
| NOTE        Windows have the attributes of:
|          window handle ·
|          window caption text, GetWindowText()
|          parent handle,      GetParent()
|          rectangle.          GetWindowRect() GetClientRect()
|          child id number.    GetDlgCtrlID()
|          Enabled or disabled. IsWindowEnabled(hwnd)
|          Active or Inactive.  GetActiveWindow() ?
|          Have focus ?        GetFocus()
|
|        Window Class attributes. WNDCLASS. GetClassInfo
|          style bit mask.
|          class name. GetClassName ?
|          module handle, Module name GetModuleFileName
|          ? cursor
|          ? icon
|          ? Menu bar resource name.
|
| In the future we want to add special controls for known classes.
| SCROLLBAR = bars may not be sub windows but part of the non client !
| BUTTON = none needed but press.
| STATIC = not needed but may label another control.
| COMBOBOX = may have scrollbars, pull down, options inside ?
| EDIT = scroll bars, new or dictated text ?
| LISTBOX
|
| We start from the bottom and work up. but previous parents are special.
| Don't duplicate the parent of current focus.
|
*/
_LOCAL BOOL ContextAddWind(HWND hwnd, int checktype)
{

        LONG Style;
        int  cwc;
        int  conType = CON_WIND;                /* default object type. */
        char szClass[MAXSTRING + 1];
        char szWndText[MAXSTRING + 1];
        PREF_FLAGS prefFlags = UserGetFlags();
```

```
if (hwnd == hwndPrvParent)
        /* We have already done with this window
        */
        return(TRUE);

/* Immediate children only.
*/
if ((checktype & CW_PARENTLEVEL) &&        ! (checktype & CW_HASFOCUS))
{
        if (hwndParent != GetParent(hwnd))
                /* Child of inactive window
                */
                return(TRUE);
}

/* Is the window iconized.
*/
Style = GetWindowLong(hwnd, GWL_STYLE);
if (Style & WS_ICONIC)
{
        conType = CON_ICON;
}

/* Is the window one of the known classes.
*/
GetClassName(hwnd, szClass, sizeof(szClass) - 1);

if (Style & WS_CHILD)
{
        /* check all control classes
        */
        for (cwc = 0; cwc < CWC_CHILD; cwc ++)
        {
                if (! lstrcmpi(szClass, szPredefClass[cwc]))
                        break;
        }
}
else
{
        /* It's popup
        */
        cwc = CWC_POPUP;
}

if (cwc == CWC_BUTTON && (Style & 0x0F) == BS_GROUPBOX)
{
        /* GroupBox is a special class
        */
        cwc = CWC_GROUPBOX;
}

/* Add children ScrollBars Control
*/
if ((prefFlags & PREF_Scroll) && cwc == CWC_SCROLLBAR)
{
        ContextAddScrollBars(hwnd, Style, cwc);
```

```
                }

                /* We must be focus or a parent of the focus to get menus and parts.
                */
                if ((checktype & CW_HASFOCUS) && (conType != CON_ICON))
                {

                        /* Does the window have a menu bar ?
                        */
                        if (
                                /* Not a child window.
                                */
                                ! (Style & WS_CHILD) &&
                                /* Already have a menu, ONLY WANT ONE.
                                */
                                ! bMenuBarExist)
                        {

                                /* Get a menu bar if there is one.
                                */
                                if ((prefFlags & PREF_Menu) && ContextAddMenu(hwnd,
GetMenu(hwnd)))
                                {
                                        bMenuBarExist = TRUE;
                                }

                        }

                        /* FOR NOW, if a popup menu is active the window is not ???
                        */
                        if (! bMenuPopExist)
                        {

                                /* Add accelerators.
                                */
                                if (bMenuBarExist && (prefFlags & PREF_Accel))
                                {
                                        ContextAddAccel(hwnd, GetMenu(hwnd));
                                }

                                /* Add contens of PMGroup
                                */
                                if (checktype == CW_HASFOCUS  && cwc == CWC_PMGROUP &&
(prefFlags & PREF_WndChild))
                                {
                                        ContextAddPMGroup(hwnd, Style);
                                }

                                /* Get system type commands.
                                */
                                if (prefFlags & PREF_SysCom)
                                {
                                        ContextAddWindSysCom(hwnd, Style);

                                }
```

```c
                                        /* Add scroll commands
                                        */
                                        if (prefFlags & PREF_Scroll)
                                        {
                                                ContextAddScrollBars(hwnd, Style, cwc);
                                        }
                        }

                        /* Add macro commands
                        */
                        if (prefFlags & PREF_Macro)
                        {

                                        /* Add non class specific macro commands only for the focus window
                                        */
                                        if (checktype == CW_HASFOCUS)
                                        {
                                                AddLngCommands(hwnd, NULL, NULL, bMenuPopExist);

                                        }

                                        /* Add windows specific macro commands for any active window
                                        */
                                        GetWindowText(hwnd, szWndText, sizeof(szWndText) - 1);
                                        AddLngCommands(hwnd, szClass, szWndText, bMenuPopExist);
                        }

                }

                /* Add the window itself after its sub parts.
                */
                if (! ContextAdd(hwnd, conType))
                        return(FALSE);
                pciLast->u.Window.cwc = cwc;

                /* We need to add window even if a user doesn't whant one
                */
                if (! (checktype & CW_HASFOCUS) &&
                        ((cwc == CWC_POPUP && ! (prefFlags & PREF_WndPopup)) ||
                        (cwc != CWC_POPUP && ! (prefFlags & PREF_WndChild))))
                {
                        /* Not valid for phrase list
                        */
                        pciLast->u.Window.bForList = FALSE;
                }
                else
                {
                        /* Valid for phrase list
                        */
                        pciLast->u.Window.bForList = TRUE;
                }

                return(TRUE);
        }

/*-------------------------------------------------------------------
```

```c
/*
| FUNCTION   _LOCAL void ContextAddPopupMenu(void)
|
| DESCRIPTION Get a popped up or selected menu or menu tree.
|
| PARAMETERS  None.
|
| RETURN     None.
|
*/
_LOCAL void ContextAddPopupMenu(void)
{
        HMENU hMenu;
        LONG Style;
        HWND hwnd = NULL;
        int iLevel = 0;

        /* Start
        */
        bMenuPopExist = FALSE;

        if (HookGet_MenuLevel() == -1)
        {
                /* No menu at all
                */
                return;
        }

        while (1)
        {
                /* Is there a menu popped up.
                */
                hMenu = HookGet_Menu(iLevel ++);
                if (hMenu == NULL)
                        /* No menu at all
                        */
                        return;

                /* Get menu from its owner window.
                ** Do  just once.
                */
                if (hwnd == NULL)
                {
                        bMenuPopExist = TRUE;
                        hwnd = HookGet_MenuWnd();
                        if (GetWindowTask(hwnd) == GetCurrentTask()) {
                                /* Don't look at Voice control
                                */
                                return;
                        }
                        Style = GetWindowLong(hwnd, GWL_STYLE);
                }

                /* If the popup menu is part of the main menu bar,
                ** then mark that we already have it.
                ** NOTE:
```

```
**  GetMenu() is undefined for WS_CHILD types.
 */
if (! (Style & WS_CHILD))
{
        if (hMenu == GetMenu(hwnd))
                bMenuBarExist = TRUE;
}

/* Add menu without accelerators
 */
if (UserGetFlags() & PREF_Menu)
{
        if (ContextAddMenu(hwnd, hMenu))
        {
                iGroupLevel++;
        }
}

/* Is it a system menu
 */
if (hMenu == GetSystemMenu(hwnd, FALSE))
{
                hwndMenuSysPop = hwnd;
}
        }
}
```

```
/*-----------------------------------------------------------------
|
| FUNCTION    BOOL CALLBACK ContextEnumProc(hwnd, lParam)
|
| DESCRIPTION Callback function that receives window handles as
|         a result of a call to the EnumWindows function.
|
| PARAMETERS  HWND hwnd - Specifies handle of the target window.
|         LONG lParam - What do we do with the data once we have it ?
|
| RETURN      Return nonzero to continue enumeration.
|
*/
BOOL FAR PASCAL ContextEnumProc(HWND hwnd, LONG lParam)
{
        return (ContextAddWind(hwnd, (int) lParam));
}
```

```
/*-----------------------------------------------------------------
|
| FUNCTION    _LOCAL char StringGetSysChar(String)
|
| DESCRIPTION Get underlined symbol fron the menu item.
|
| PARAMETERS  PSTR String - Specifies menu string.
|
| RETURN      Underlined symdol.
|
*/
```

```c
_LOCAL char StringGetSysChar(PSTR String)
{
        while (*String)
        {
                if (*(String++) == '&')
                {
                        /* We have found &
                        */
                        break;
                }
        }
        /* Return address of the next one
        */
        return(*String);
}


/*------------------------------------------------------------------
|
| FUNCTION    _LOCAL int ContextPakWind(hwnd)
|
| DESCRIPTION Pak a string description for the window type object.
|        User pciLast to identify the object.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|
| RETURN     Length of the caption text.
|
*/
_LOCAL int ContextPakWind(HWND hwnd)
{
        int  len;

        /* If window not active then ignore it.
        */
        if (
                (! IsWindowEnabled(hwnd)
                || ! IsWindowVisible(hwnd)))            /* Not really working ??? */
                return(0);

        /* What is its caption text ?
        */
        len = GetWindowText(hwnd, szCaptionBuf, sizeof(szCaptionBuf) - 1);

        /*
        ** What is its class.
        */
        switch (pciLast->u.Window.cwc)
        {
                case CWC_EDIT:
                case CWC_COMBOBOX:
                case CWC_LISTBOX:
                case CWC_SCROLLBAR:
                        /* Edit/Comb/List captions are the current text inside them ?
                        */
                        len = 0;
                        break;
```

```
                  ·  case CWC_GROUPBOX:
                     case CWC_STATIC:
                             /* If static or group box has & it lable something
                             */
                             if (! StringGetSysChar(szCaptionBuf))
                             {
                                     len = 0;
                             }
                             break;

                     default:
                             :

            }
            return(len);
}

/*-----------------------------------------------------------------------
|
| FUNCTION    _LOCAL int ContextPakMenu(hMenu, idItem, fuFlags)
|
| DESCRIPTION Get an option from a menu.
|
| PARAMETERS  HMENU hMenu  - Specifies handle to the menu.
|         int idItem   - Specifies item ID.
|         UINT fuFlags - Specifies item flags.
|
| RETURN      Length of the caption text.
|
| NOTE        When sys menus of child windows are popped up:
|         they have a popup menu type with a caption of junk ?
|         The high MF_ values str not valid for MF_POPUP or menu bars.
|         high = the number of entries in the popup.
|
*/
_LOCAL int ContextPakMenu(HMENU hMenu, int idItem, UINT fuFlags)
{
        WORD State;
        int  len = 0;

        if (hMenu == NULL) return(0);

        State = GetMenuState(hMenu, idItem, fuFlags);
        if (State == -1) return(0);

        /* Is the item available grayed, disabled ?
        ** -1 == not exist.
        */
        if ((State & MF_DISABLED )
                ||(State & MF_GRAYED  ))
                return 0;

        if (! (State & MF_POPUP))
        {
                if ((State & MF_BITMAP)
```

```
                              || (State & MF_OWNERDRAW))
                          return 0;;
              }

              /* Get the text description.
              */
              len = GetMenuString(hMenu, idItem, szCaptionBuf, sizeof(szCaptionBuf) - 1, fuFlags);

              return(len);
}


/*----------------------------------------------------------------
 |
 | FUNCTION   _LOCAL int ContextPakSysCom(hwnd, iSysCom)
 |
 | DESCRIPTION Create system command string.
 |
 | PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
 |        int iSysCom - SC_...
 |
 | RETURN     Length of the caption text.
 |
 */
_LOCAL int ContextPakSysCom(HWND hwnd, int iSysCom)
{
        char Str[MAXSTRING + 1];
        int  len  = 0;

        switch (iSysCom)
        {
                case SC_KEYMENU:
                case SC_MOUSEMENU:
                        /* We can get other options by pulling down the sys menu.
                        */
                        len = wsprintf(
                                szCaptionBuf,
                                "%s %s",
                                (LPSTR)UserGetDefWord((GetWindowLong(hwnd,GWL_STYL
E) & WS_CHILD) ? IDW_CHILD : IDW_POPUP),
                                (LPSTR)UserGetDefWord(IDW_SYSMENU));
                        break;

                case SC_CLOSE:                    /* May be close window or app. */
                case SC_MINIMIZE:
                case SC_MAXIMIZE:
                case SC_RESTORE:
                        /* List these visible controls seperately.
                        */
                default:
                        GetMenuString(GetSystemMenu(hwnd, FALSE), iSysCom, Str,
MAXSTRING, MF_BYCOMMAND);
                        len = StringClip(Str);
                        if (len)
                        {
                                len = wsprintf(
                                        szCaptionBuf,
```

```
                                        "%s %s",
                                        (LPSTR)Str,
                                        (LPSTR)UserGetDefWord((GetWindowLong(hwnd,GWL
_STYLE) & WS_CHILD) ? IDW_CHILD : IDW_POPUP));
                        }
        }

        return(len);

}

/*-----------------------------------------------------------------------
|
| FUNCTION    _LOCAL int ContextPakScroll(iScrlCom)
|
| DESCRIPTION Create scroll command string.
|
| PARAMETERS  int iScrlCom - Specifies scroll command.
|
| RETURN      Length of the caption text.
|
*/
_LOCAL int ContextPakScroll(int iScrlCom)
{
        int len;
        int idWord;

        /* First try all type of horizontall scroll
        */
        if (iScrlCom & SCRLS_HORZ)
        {
                switch (pciLast->u.ScrlCom & SCRLS_ACT)
                {
                        case SB_LINEUP:
                                /* line left
                                */
                                idWord = IDW_LINELEFT;
                                break;
                        case SB_LINEDOWN:
                                /* line right
                                */
                                idWord = IDW_LINERIGHT;
                                break;
                        case SB_PAGEUP:
                                /* page left
                                */
                                idWord = IDW_PAGELEFT;
                                break;
                        case SB_PAGEDOWN:
                                /* page right
                                */
                                idWord = IDW_PAGERIGHT;
                                break;
                }
        }
        /* Now all type of vertical scroll
```

```
    |
    */
_LOCAL int ContextPak(void)
{
        int  len;
        HWND hwnd = pciLast->hwnd;

        *szCaptionBuf = '\0';

        switch (pciLast->conType)
        {

                case CON_WIND:
                case CON_ICON:
                        /* Does the user want to have window names ?
                        */
                        if (! pciLast->u.Window.bForList)
                        {
                                len = NULL;
                                break;
                        }
                        /* Does alias name exist ?
                        */
                        if (pciLast->u.Window.szName)
                        {
                                lstrcpy(szCaptionBuf, pciLast->u.Window.szName);
                                len = lstrlen(szCaptionBuf);
                        }
                        /* Try to get caption
                        */
                        else
                        {
                                len = ContextPakWind(hwnd);
                        }
                        break;

                case CON_SYSCOM:
                        /* The system command for the window.
                        */
                        len = ContextPakSysCom(hwnd, pciLast->u.SysCom);
                        break;

                case CON_SCROLL:
                        len = ContextPakScroll(pciLast->u.ScrlCom);
                        break;

                case CON_MENU:
                        /* Does alias name exist ?
                        */
                        if (pciLast->u.Menu.szName)
                        {
                                lstrcpy(szCaptionBuf, pciLast->u.Menu.szName);
                                len = lstrlen(szCaptionBuf);
                        }
                        /* Get an item from a popped up menu.
                        */
```

```
                             else {
                                     len = ContextPakMenu(pciLast->u.Menu.hMenu,          pciLast-
>u.Menu.id,    MF_BYCOMMAND);
                             }
                             break;

               case CON_MENUPOPUP:
                             /* Read an item from the menu bar.
                             */
                             len = ContextPakMenu(pciLast->u.MenuPop.hMenu, pciLast-
>u.MenuPop.iEntry, MF_BYPOSITION);
                             break;

               case CON_ACCEL:
                             /* Accelerator has the same text as a menu item (it available thought)
                             */
                             len = GetMenuString(pciLast->u.Acc.hMenu, pciLast->u.Acc.id,
szCaptionBuf,
                                     sizeof(szCaptionBuf) - 1, MF_BYCOMMAND);
                             break;

               case CON_LAUNCH :
                             /* PM item title
                             */
                             lstrcpy(szCaptionBuf, pciLast->u.PMItem.szTitle);
                             len = lstrlen(szCaptionBuf);
                             break;

               case CON_MACRO:
                             /* Macro name
                             */
                             lstrcpy(szCaptionBuf, (pciLast->u.pMacro)->szName);
                             len = lstrlen(szCaptionBuf);
                             break;

               default: return(0);
       }

       /* Chop out the ampersands (&) and tabs.
       */
       if (len)
               len = StringClip(szCaptionBuf);
       else
               *szCaptionBuf = '\0';

       if (len > iCaptionLen)
               iCaptionLen = len;

#ifdef DEBUG_DLG
       /* Pack debug info
       */
       if (DebugFlag & DEBUG_ContFull)
       {
               len = ContextPakDebug();
               if (len > iCaptionLen) iCaptionLen = len;
       }
```

```c
#endif

            /* Return length of the string
            */
            return(len);

}


/*-----------------------------------------------------------------------
|
| FUNCTION    BOOL ContextCheck(bPrefChange)
|
| DESCRIPTION Hook the context window to the status window.
|
| PARAMETERS  BOOL bPrefChange - Rebuild list anyway
|
| RETURN    TRUE = A change in the context ?
|
| NOTE      This is called every so often to check for context changes.
|           Watch for the change in focus thru the hook routines ?
|           Menus don't change the focus ! we must watch messages for them !
|           When we select an icon the focus = null the active window is icon.
|
*/
BOOL ContextCheck(BOOL bPrefChange)
{

        int     checktype;
        int     changetype;
        unsigned PrevCheckSum;

        changetype = HookGet_Change();

        /* Does anything change ?
        */
        if (changetype == HCHANGE_NONE && ! bPrefChange)
                return(FALSE);

        /* Set up to enumerate the windows.
        */
        if (lpprocContext == NULL)
        {
                lpprocContext = MakeProcInstance(ContextEnumProc, VChInst);
        }

        /* First we check context save options (when old focus valid).
        */
        if (GetWindowTask(GetActiveWindow()) == GetCurrentTask())
        {

            if (IsWindow(hwndFocus)         && (! IsIconic(hwndActive) || hwndActive ==
hwndFocus))
                {
                        /* Context still good for now, but we need to check preferences
                        */
```

```c
                        if (! bPrefChange)
                        {
                                return(FALSE);
                        }
                }
                else
                {
                        /* We cannot find our active window.
                        ** Don't look to it.
                        */
                        hwndFocus = 0;
                }

        }
        else
        {
                /* Who is active now.
                */
                hwndActive = GetActiveWindow();

                /* Who has focus right now.
                */
                hwndFocus = GetFocus();

                /* We should start
                */
                if (! hwndFocus)
                        hwndFocus = hwndActive;
        }


        /*
        ** restart the context list.
        */
        PrevCheckSum = iCheckSum;        /* Save the previous to compare. */
        ContextListInit();

        /*
        ** Check for a pop up menu active.
        ** ALWAYS highest focus priority.
        */
        ContextAddPopupMenu();

        if (hwndFocus)
        {
                /*
                ** Get those windows that are children of the current focus.
                ** NOTE: Items in the immediate focus should be on top !
                ** Move up the hierarchy to the modal level or the non WS_CHILD ?
                */

                hwndParent = hwndFocus;
                hwndPrvParent = NULL;
                checktype = 0;

                while (hwndParent != NULL)
```

```
                    {
                            if (! IsWindowEnabled(hwndParent))  /* The previous was top. */
                                    break;

                            if (! IsIconic(hwndParent))
                            {
                                    EnumChildWindows(hwndParent, lpprocContext, checktype);
                                    iGroupLevel ++;
                            }

                            /*
                            ** Store the parent level. (May not be a real option.)
                            */
                            ContextAddWind(hwndParent, CW_HASFOCUS | checktype);
                            hwndPrvParent = hwndParent; /* Don't duplicate in siblings. */
                            iGroupLevel ++;
                            checktype = CW_PARENTLEVEL;

                            /*
                            ** Break after Active window
                            */
                            if (hwndParent == hwndActive)
                            {
                                    break;
                            }

                            /*
                            ** Does it have a parent ?
                            */
                            hwndParent = GetParent(hwndParent);
                    }

            }

    /*
    ** Get other applications. except if someone above is system modal.
    ** WS_OVERLAPPED and WS_POPUP type windows.
    */
    EnumWindows(lpprocContext, 0);

    ContextAdd(NULL, 0);                            /* Checksum the last. */

    return(PrevCheckSum != iCheckSum || changetype > HCHANGE_POSSIBLE);

}

/*------------------------------------------------------------------
|
| FUNCTION    void ContextListAdd(void)
|
| DESCRIPTION Build a list of siblings and children.
|
| PARAMETERS  None.
|
| RETURN      None.
|
```

```c
*/
void ContextListAdd(void)
{
        int len;
        int iEntry = 0;

        ContextCheck(FALSE);                    /* One final check before packing. */

        iCaptionLen  = 13;                      /* Minimum size. */

#ifdef DEBUG_DLG
        iDebugCapLen = 0;
#endif

        for (pciLast = pciFirst; pciLast != NULL; pciLast = pciLast->pciNext, iEntry ++)
        {
                len = ContextPak();
                if (! len) continue;

                /* Send a message adding the window caption to the list
                ** in the dialog.
                */
                if (! PhraseListAdd(szCaptionBuf, iEntry)) break;
        }

#ifdef DEBUG_DLG
        /* Set the tabs and columns.
        */
        if (DebugFlag & DEBUG_ContFull)
        {

                ContextTabs[0] = (iCaptionLen + 4) * 10;
                ContextTabs[1] = (iCaptionLen + 12) * 10;
                ContextTabs[2] = iCaptionLen + 16 + iDebugCapLen;

        }
#endif
}

/*-------------------------------------------------------------------
|
| FUNCTION    void ContextListSelect(iEntry)
|
| DESCRIPTION The user selected a word from the list.
|         Take some default MACRO action based on the context type
|
| PARAMETERS  int iEntry - Specifies numer of list item;
|
| RETURN      None.
|
*/
void ContextListSelect(int iEntry)
{
        HWND   hwnd;
        MACRO  macro;
```

```c
if (iEntry < 0) return;

/*
** Find the window in the list.
*/
for (pciLast = pciFirst; iEntry; iEntry --)
{
        if (pciLast == NULL)
                return;         /* THIS SHOULD NEVER HAPPEN */
        pciLast = pciLast->pciNext;
}
hwnd = pciLast->hwnd;

/* We keep focus and it valid.
*/
if (GetWindowTask(GetActiveWindow()) == GetCurrentTask())
{
        SetFocus(hwndFocus);
}

/* Default macros are to be executed on hwnd.
*/
macro.szWndClass = NULL;
macro.szDesc    = NULL;
macro.pNext     = NULL;

switch (pciLast->conType) {

        case CON_SYSCOM:
                /* A system command from the system command menu to the window.
                ** PostMessage(hwnd, WM_SYSCOMMAND, iEntry, NULL);
                */
                macro.cmdType       = CMD_SYSTEM;
                macro.Cmd.System.wCmd = pciLast->u.SysCom;
                break;


        case CON_SCROLL:
                /* PostMessage
                */
                macro.cmdType       = CMD_MESSAGE;
                macro.Cmd.Msg.wMsg = (pciLast->u.ScrlCom & SCRLS_HORZ) ?
WM_HSCROLL : WM_VSCROLL;
                macro.Cmd.Msg.wParam = pciLast->u.ScrlCom & SCRLS_ACT;

                if (pciLast->u.ScrlCom & SCRLS_WIN)
                {
                        macro.Cmd.Msg.lParam = MAKELONG(0, hwnd);
                        hwnd = GetParent(hwnd);
                }
                else
                {
                        macro.Cmd.Msg.lParam = 0L;
                }
                break;
```

```
case CON_ICON:
        /* Restore the iconic window.
        ** NOTE:
        ** Iconic windows don't get focus. they just activate.
        ** OpenIcon(hwnd);
        */
        macro.cmdType       = CMD_SYSTEM;
        macro.Cmd.System.wCmd = SC_RESTORE;
        break;


case CON_WIND:
        if ((pciLast->u.Window.cwc == CWC_STATIC) || (pciLast-
>u.Window.cwc == CWC_GROUPBOX))
                {
                GetWindowText(hwnd, szCaptionBuf, sizeof(szCaptionBuf) - 1);
                macro.cmdType       = CMD_KEY;
                macro.Cmd.Key.cKey = (char)
VkKeyScan(StringGetSysChar(szCaptionBuf));
                macro.Cmd.Key.AltPressed = (BYTE) 1;
                macro.Cmd.Key.ShiftPressed = (BYTE) 0;
                macro.Cmd.Key.CtrlPressed =  (BYTE) 0;
                }
        else
                {
                /* Choose the window as the current window.  For top level
windows this

                ** will result in their being activated.  For items in dialog boxes
                ** this will result in their being selected.
                */
                macro.cmdType = CMD_SELECT;
                }
        break;


case CON_MENUPOPUP:
        /* An item on the windows menu bar.
        ** Pull down the popup menu.
        */
        macro.cmdType        = CMD_MENUPOPUP;
        macro.Cmd.MenuPopup.iKeyPos = pciLast->u.MenuPop.iKeyPos;

        if (GetMenu(hwnd) == pciLast->u.MenuPop.hMenu)
                macro.Cmd.MenuPopup.wLevel = 0;
        else
                macro.Cmd.MenuPopup.wLevel = 1;
        break;


case CON_MENU:
        /* A menu item in the active menu.
        ** Execute the menu item.
        ** PostMessage(hwnd, WM_COMMAND, iEntry, NULL);
        */
```

```c
            if (hwndMenuSysPop)
            {
                    /* Menu item chosen from system menu.
                    */
                    macro.cmdType        = CMD_SYSTEM;
                    macro.Cmd.System.wCmd = pciLast->u.Menu.id;
            }
            else
            {
                    /* Menu item chosen from the menu bar.
                    */
                    macro.cmdType        = CMD_MENU;
                    macro.Cmd.Menu.id = pciLast->u.Menu.id;
            }
            break;

    case CON_ACCEL:
            /* Accelerator key
            */
            macro.cmdType        = CMD_MENU;
            macro.Cmd.Menu.id = pciLast->u.Acc.id;
            break;

    case CON_LAUNCH:
            /* Just execute
            */
            macro.cmdType     = CMD_LAUNCH;
            macro.szDesc      = pciLast->u.PMItem.szFile;
            break;

    case CON_MACRO:
            macro.cmdType    = pciLast->u.pMacro->cmdType;
            macro.Cmd        = pciLast->u.pMacro->Cmd;
            macro.itemid     = pciLast->u.pMacro->itemid;
            macro.szDesc     = pciLast->u.pMacro->szDesc;
            break;

    default :
            return;
}

VCM_Execute(&macro, hwnd);

}
```

```
/*
** File: HOOK.C
**
** Module for Hooking Window's queue and tracking relevant messages.
**
** Interface functions: HookGet_Change
**               HookGet_Menu
**               HookGet_MenuAtLevel
**               HookGet_MenuLevel
**               HookGet_MenuWnd
**               HookInstall
**               HookJournalBusy
**               HookFreeJournal
**               Record
**
** Exported functions:  HookMain
**               HookGetMsgProc
**               HookSndMsgProc
**               PlayProc           ,
**               RecProc
**
** Private functions:  HookMenuClear
**               HookMessage
**               PlayNotify .
**               RecNotify
**
**
***************************************************************************
**************************************************************************/

#include <windows.h>
#include "vtools.h"

typedef struct
{        // Another message type
        DWORD IParam;                    /* This was backwards before ? */
        WORD  wParam;
        WORD  wMsg;
        HWND  hWnd;
} CALLWNDPROC;            /* NOTE: Parameters are oposite of LPMSG ? */

typedef CALLWNDPROC FAR *LPCALLWNDPROC;


/*---------------------------------------------------------------
|
| Module local variables.
|
*/
HANDLE hInst;           // Instance Handle given in LibMain()
HHOOK  hGetMsgHook;     // Handle to the getmessage hook
HHOOK  hSndMsgHook;     // Handle to the callwndproc hook
HHOOK  hJournalHook;    // Current journal record/playback hook function .

/*---------------------------------------------------------------
|
| --- Variables for Playback ---
|
```

```c
*/
static LPRECORD  lpJmlList;     // Handle to the list of journal events
static BOOL    bJournalBusy;    // Is the DLL busy recording or playing back?
static DWORD   dwlnitPlaybackTime; // Initial time of Playback() call
static short   sPlaybackSpeed;   // Speed given to Playback() (0 or -1)
static DWORD   dwPrevMsgTime;    // Time of previously played back event

static HWND    hWndNotify;
static UINT    wMsgNotify;
static UINT    wStopKey;
static UINT    wMouRec;

/*------------------------------------------------------------------------
|
| --- Context manager tracking. ---
|
*/
static int    Hook_Change;            /* context change type. */
static HWND    Hook_MenuhWnd;          /* The window owning the menu. */
static int    Hook_MenuLevel;         /* The menu stack level. -1=none */
static HMENU   Hook_MenuSelect;  /* Selected item from the current level. */

static enum
{
        /*
        ** If we are tracking a multi message operation.
        */
        HT_NONE,  /* Watch for nothing. */
        HT_ACCEL, /* Watch for an accelerator key press. */
} Hook_Track;

#define MENUSTACKQTY 6              /* How many sub levels to store. */

static HMENU   Hook_MenuStack[MENUSTACKQTY];  /* currently active menu. */


/*------------------------------------------------------------------------
|
| FUNCTION   int CALLBACK HookMain(hinst, wDataSeg, wHeapSize, lpszCmdLine)
|
| DESCRIPTION Part of the LibMain that belongs to the hook system.
|
| PARAMETERS  HINSTANCE hinst  - Identifies the instance of the DLL.
|        WORD wDataSeg   - Specifies the value of the data
|                   segment (DS) register.
|        WORD wHeapSize   - Specifies the size of the heap defined
|                   in the module-definition file.
|        LPSTR lpszCmdLine - Points to a null-terminated string
|                   specifying command-line information.
|
| RETURN     1 if it is successful. Otherwise, it should return 0.
|
*/
int CALLBACK HookMain(HINSTANCE hinst, WORD wDataSeg, WORD wHeapSize, LPSTR
lpszCmdLine)
{
```

```c
        hInst = hinst;
        bJournalBusy  = FALSE;
        hGetMsgHook   = NULL;
        hSndMsgHook   = NULL;

        Hook_Change   = HCHANGE_NONE;
        Hook_MenuLevel = -1;
        Hook_Track    = HT_NONE;

        return (TRUE);
}

/*-------------------------------------------------------------------
|
| FUNCTION    int WINAPI HookGet_Change(void)
|
| DESCRIPTION Has part of the context changed.
|         Because looking for changes is not an exact science we know some
|         events are always a change and some are just possible.
|         Keep 2 flags.
|
| PARAMETERS  None.
|
| RETURN      Hook change status.
|
*/
int WINAPI HookGet_Change(void)
{
        int Prev;

        Prev = Hook_Change;
        Hook_Change = HCHANGE_NONE;

        return(Prev);
}

/*-------------------------------------------------------------------
|
| FUNCTION    HMENU WINAPI HookGet_Menu(level)
|
| DESCRIPTION Return the handle to the current popped up menu.
|
| PARAMETERS  int level - the inverse of the menu stack level. 0=top-most
|
| RETURN      NULL = no menu is popped up
|
*/
HMENU WINAPI HookGet_Menu(int level)
{
        if (level > Hook_MenuLevel) return(NULL);

        return(Hook_MenuStack[Hook_MenuLevel - level]);
}

/*-------------------------------------------------------------------
```

```
/*
|  FUNCTION · HMENU WINAPI HookGet_MenuAtLevel(level)
|
|  DESCRIPTION Return the handle to the menu at the given level.
|
|  PARAMETERS  int level - the menu stack level. 0=top-most
|
|  RETURN     NULL = no menu is popped up.
|
*/
HMENU WINAPI HookGet_MenuAtLevel(int level)
{
        if (level > Hook_MenuLevel) return(NULL);

        return(Hook_MenuStack[level]);
}

/*-----------------------------------------------------------------
|
|  FUNCTION    int WINAPI HookGet_MenuLevel()
|
|  DESCRIPTION Return the menu level.
|
|  PARAMETERS  None.
|
|  RETURN     The menu level : NULL = no menu is popped up.
|
*/
int WINAPI HookGet_MenuLevel()
{
        return(Hook_MenuLevel);
}

/*-----------------------------------------------------------------
|
|  FUNCTION    HWND WINAPI HookGet_MenuWnd(void)
|
|  DESCRIPTION Returns the owner of the popped up window.
|          Only valid if there IS a popped up menu !
|
|  PARAMETERS  None.
|
|  RETURN     Handle to the window.
|
*/
HWND WINAPI HookGet_MenuWnd(void)
{
        return(Hook_MenuhWnd);
}

/*-----------------------------------------------------------------
|
|  FUNCTION    static void HookMenuClear(void)
|
|  DESCRIPTION Clear menu toggles.
|
```

```
|  PARAMETERS  None.
|
|  RETURN     None.
|
*/
static void HookMenuClear(void)
{
        if (Hook_MenuLevel == -1) return;
        Hook_MenuLevel = -1;                    /* No popup menu. */
        Hook_Change |= HCHANGE_DEFINATE;
}


/*-------------------------------------------------------------------------
|
|  FUNCTION   static void PASCAL HookMessage(hWnd, wMsg, wParam, lParam)
|
|  DESCRIPTION Check for common context indication messages.
|          Use command message checker for PostMessage and SendMessage
|          because we never really know which will be used.
|
|  PARAMETERS  HWND hWnd   - Specifies the handle of the window
|          UINT wMsg   - Specifies the message
|          WORD wParam - Specifies 16 bits of additional
|                  message-dependent information
|          LONG lParam - Specifies 16 bits of additional
|                  message-dependent information
|
|  RETURN     None.
|
*/
static void PASCAL HookMessage(HWND hWnd, UINT wMsg, WORD wParam, LONG lParam)
{

        switch (wMsg)
        {

                /*
                ** Menu level tracking.
                */

                case WM_INITMENU:
                        /*
                        ** The bottom level menu is initialized.
                        */
                        Hook_MenuhWnd  = hWnd;
                        Hook_MenuLevel = -1;
                        Hook_MenuSelect = NULL;
                        Hook_Track = HT_NONE;
                        Hook_Change |= HCHANGE_DEFINATE;
                        break;


                case WM_INITMENUPOPUP:
                        /*
                        ** The menu will pop up onto the screen.
                        ** NOTE: The context manager needs this to tell if a menu is up.
```

```c
*/
                if (Hook_MenuSelect == wParam)
                {
                        if (Hook_MenuLevel >= MENUSTACKQTY-1) break;           /*
SORRY */
                        Hook_MenuLevel ++;
                }
                else
                {
                        /*
                        ** NOTE:
                        ** Of the Popup is initialized without having selected it
                        ** then it is not a normal menu popup ? What do i do ?
                        ** NOTE:
                        ** This works for custom popups.
                        */
                        Hook_MenuLevel = 0;            /* Don't know where this is from ?
*/
                        Hook_Track = HT_ACCEL;
                }

                Hook_MenuSelect = NULL;
                Hook_MenuStack[Hook_MenuLevel] = wParam;
                Hook_Change |= HCHANGE_DEFINATE;
                break;

        case WM_MENUSELECT:
                /*
                ** Watch for the pop up menu being removed.
                ** or the select being moved.
                ** wParam = the item seelcted, (handle if popup)
                ** HIWORD(lParam) = our parent.
                */

                if (wParam == 0 && lParam == 0xFFFFL)
                {
                        HookMenuClear();
                        break;
                }
                if (Hook_MenuLevel == -1)
                {
                        Hook_MenuStack[++ Hook_MenuLevel] = HIWORD(lParam);
                        Hook_Change |= HCHANGE_DEFINATE;
                }
                else
                {
                        if (HIWORD(lParam) == Hook_MenuSelect)
                        {
                                /*
                                ** NOTE:
                                ** This occurs if the menu select is moved back to the
parent-
                                ** But the child is left on the screen ?
                                */
                                Hook_MenuLevel ++;                          /* same as
last. */
```

```
                                    Hook_Change |= HCHANGE_DEFINATE;
                    }
                    else
                    {
                            while (Hook_MenuLevel > 0)
                            {
                                    if (HIWORD(lParam) ==
Hook_MenuStack[Hook_MenuLevel])
                                            break;
                                    Hook_MenuLevel --;
                                    Hook_Change |= HCHANGE_DEFINATE;
                            }
                    }
            }

            Hook_Track = HT_NONE;
            Hook_MenuSelect = wParam;
            break;

    case WM_SYSCOMMAND:
            /*
            ** Check for the window being maximized, minimized or restored.
            */
            switch (wParam)
            {
                    case SC_MAXIMIZE :
                    case SC_MINIMIZE :
                    case SC_RESTORE  :
                            Hook_Change |= HCHANGE_DEFINATE;
                            break;
            }

    case WM_COMMAND:
            /*
            ** Clear the menu if present.
            ** NOTE: Accelerator keys only exit with a WM_COMMAND
            */
            if (Hook_Track == HT_ACCEL)
                    HookMenuClear();
            break;

    case WM_ACTIVATEAPP:
            /*
            ** We are changing applications.
            */
            Hook_Change |= HCHANGE_TASK;
            break;

    case WM_ACTIVATE:
            /*
            ** The window activation is changing. similar to focus.
            */
    case WM_SETFOCUS:
    case WM_KILLFOCUS:
            /*
```

```
                    ** The focus is changing.
                    */
                    Hook_Change |= HCHANGE_POSSIBLE;
                    break;

            case WM_SETTEXT:
                    /*
                    ** Some text is being set to a window or control.
                    ** Most likely it is a change.
                    */
                    Hook_Change |= HCHANGE_DEFINATE;
                    break;

            case WM_SHOWWINDOW:
                    Hook_Change |= HCHANGE_DEFINATE;
                    break;

            case WM_CREATE:
                    /*
                    ** The window is created.
                    */
            case WM_PAINT:
            case WM_NCPAINT:
            case WM_NCCALCSIZE:
            case WM_CTLCOLOR:
            case WM_ENTERIDLE:
                    /*
                    ** NOTE: It could be (Not necesssary) a change.
                    */
                    Hook_Change |= HCHANGE_POSSIBLE;
                    break;

        }
}

/*-----------------------------------------------------------------------
|
| FUNCTION    DWORD CALLBACK HookGetMsgProc(nCode, wParam, lpMsg)
|
| DESCRIPTION The HookGetMsgProc function is a callback function that
|             the system calls whenever the GetMessage function has
|             retrieved a message from an application queue.
|             The system passes the retrieved message to the callback
|             function before passing the
|             message to the destination window procedure.
|
| PARAMETERS  int nCode    - Specifies whether the callback function
|                            should process the message or call the
|                            CallNextHookEx function. If this parameter is
|                            less than zero, the callback function should
|                            pass the message to CallNextHookEx without
|                            further processing.
|             WORD wParam  - Specifies a NULL value.
|             LPMSG lpMsg  - Points to an MSG structure that contains
|                            information about the message.
|
```

```
| RETURN    The callback function should return zero.
|
*/
DWORD CALLBACK HookGetMsgProc(int nCode, WORD wParam, LPMSG lpMsg)
{
        if (nCode == HC_ACTION)
        {
                HookMessage(lpMsg->hwnd, lpMsg->message, lpMsg->wParam, lpMsg-
>lParam);
                if (lpMsg->message == WM_MOUSEMOVE)
                {
                        lpMsg->wParam &= ~MK_MBUTTON;
                }
        }

        return CallNextHookEx(hGetMsgHook, nCode, wParam, (LONG)lpMsg);
}

/*-----------------------------------------------------------------
|
| FUNCTION   DWORD CALLBACK HookSndMsgProc(nCode, wParam, lpMsg)
|
| DESCRIPTION Hooks all SendMessage calls.
|
| PARAMETERS  int nCode        -Specifies whether the callback function
|                        should process the message or call the
|                        CallNextHookEx function. If this parameter
|                        is less than zero, the callback function
|                        should pass the message to CallNextHookEx
|                        without further processing.
|        WORD wParam        -Specifies whether the message is sent by
|                        the current task. This parameter is
|                        nonzeroif the message is sent;
|                        otherw ise, it is NULL.
|        LPCALLWNDPROC lpMsg -Points to a structure that contains
|                        details about the message.
|
| RETURN     The callback function should return zero.
|
*/
DWORD CALLBACK HookSndMsgProc(int nCode, WORD wParam, LPCALLWNDPROC lpMsg)
{
        if (nCode == HC_ACTION)
        {
                HookMessage(lpMsg->hWnd, lpMsg->wMsg, lpMsg->wParam, lpMsg->lParam);
        }
        return CallNextHookEx(hSndMsgHook, nCode, wParam, (LONG)lpMsg);
}

/*-----------------------------------------------------------------
|
| FUNCTION    void WINAPI HookInstall(fInstall)
|
| DESCRIPTION Set up all neccessary hooking code to view all messages.
|
| PARAMETERS  BOOL fInstall - Specifies install/uninstall toggle.
```

```
|
| RETURN    None.
|
*/
void WINAPI HookInstall(BOOL fInstall)
{
        if (fInstall)
        {  // Install only if there isn't already a hook installed
                /*
                ** Install hook for posted messages.
                */
                if (!hGetMsgHook)
                        hGetMsgHook = SetWindowsHookEx(WH_GETMESSAGE,
(FARPROC)HookGetMsgProc, hInst, NULL);

                /*
                ** Install hook for sent messages.
                */
                if (!hSndMsgHook)
                        hSndMsgHook = SetWindowsHookEx(WH_CALLWNDPROC,
(FARPROC)HookSndMsgProc, hInst, NULL);
        }
        else
        {
                UnhookWindowsHookEx(hGetMsgHook);
                UnhookWindowsHookEx(hSndMsgHook);
                hGetMsgHook = NULL;
                hSndMsgHook = NULL;
        }
}

/*-----------------------------------------------------------------
|
| FUNCTION    BOOL WINAPI HookJournalBusy(void)
|
| DESCRIPTION Return whether or not the DLL has a journal hook already
|          installed
|
| PARAMETERS  None.
|
| RETURN      TRUE if journal busy.
|
*/
BOOL WINAPI HookJournalBusy(void)
{
        return bJournalBusy;  // Is journal playback active?
}


/*-----------------------------------------------------------------
|
| FUNCTION    static void PlayNotify(void)
|
| DESCRIPTION Notify about end of playyback.
|
| PARAMETERS  None.
|
```

```
|  RETURN     None.
|
*/
static void PlayNotify(void)
{

        if (hWndNotify)
        {
                SendMessage(hWndNotify, wMsgNotify, 0, 0L);
        }

}

/*-------------------------------------------------------------------
|
|  FUNCTION   DWORD CALLBACK PlayProc(nCode, wParam, lpMsg)
|
|  DESCRIPTION The PlayProc function is a callback function that
|             a library can use to insert mouse and keyboard messages into
|             the system message queue.
|
|  PARAMETERS  int nCode       - Specifies whether the callback function
|                                should process the message or call the
|                                CallNextHookEx function. If this parameter
|                                is less than zero, the callback function
|                                should pass the message to CallNextHookEx
|                                without further processing. `
|             WORD wParam      - Specifies a NULL value.
|             LPEVENTMSG lpMsg - Points to an EVENTMSG structure that
|                                represents the message being processed
|                                by the callback function.
|
|  RETURN     The callback function should return a value that represents
|             the amount of time, in clock ticks, that the system should
|             wait before processing the message. This value can be computed
|             by calculating the difference between the time members of the
|             current and previous input messages. If the function returns
|             zero, the message is processed immediately.
|
*/
DWORD CALLBACK PlayProc(int nCode, WORD wParam, LPEVENTMSG lpMsg)
{
        DWORD   dwRetcode = NULL;
        BOOL    bCallNext = TRUE;
        LPRECORD lpList;

        switch (nCode)
        {
                case HC_SKIP :
                        // See if we are all done playing back
                        if (!lpJrnlList)
                        {
//OutputDebugString("HC_SKIP - Next event is NULL so we're all done.\n");
                                UnhookWindowsHookEx(hJournalHook);
                                PlayNotify();
                                bJournalBusy = FALSE;
```

```
//      if (!wNumEvents)
//        OutputDebugString(" Played the number of events recorded.\n");
                    }
                else {
//      wNumEvents--;

                            lpList = lpJmlList->pNext;
                            Gfree(lpJmlList);
                            lpJmlList = lpList;
                    }
                    bCallNext = FALSE;
                    break;


            case HC_GETNEXT :
                // Lock and playback this member of the list.

                if (lpJmlList)
                {

                            lpMsg->message = lpJmlList->msg.message;
                            lpMsg->paramL = lpJmlList->msg.paramL;
                            lpMsg->paramH = lpJmlList->msg.paramH;

                            switch (sPlaybackSpeed)
                            {
                                case -1 :  // Full Speed
                                        lpMsg->time = GetTickCount();
                                        dwRetcode = dwInitPlaybackTime -
GetTickCount() + GetDoubleClickTime() + 1;

                                        if ((long)dwRetcode < 0) // if time has gone by
return

                                                dwRetcode = 0;
                                // 0 for the wait time.
                                        break;

                                default :
                                case 0 :        // Original Speed
                                        lpMsg->time = lpJmlList->msg.time +
dwInitPlaybackTime;

                                        dwRetcode = lpMsg->time - GetTickCount();
                                        if ((signed long)dwRetcode < 0) // if time has
gone by return

                                                dwRetcode = 0;
                                // 0 for the wait time.
                                        break;
                            }

                }
                bCallNext = FALSE;
                break;


            case HC_SYSMODALON :
                // A system modal dialog box has appeared.
                // Something bad must have happened.
```

```c
                        // Free all remaining event structures and unhook.

                        // Should some sort of error message be displayed to the user when
                        // we receive the HC_SYSMODALOFF to say that we stopped playback?

                        while (lpJmlList)
                        {
                                lpList = lpJmlList->pNext;
                                Gfree(lpJmlList);
                                lpJmlList = lpList;
                        }

                        UnhookWindowsHookEx(hJournalHook);
                        PlayNotify();
                        bJournalBusy = FALSE;
                        break;

                default :
                        break;
                }

        if (bCallNext)
        {
                dwRetcode = CallNextHookEx(hJournalHook, nCode, wParam, (LONG)lpMsg);
        }

        return dwRetcode;
}

/*------------------------------------------------------------------
|
| FUNCTION    void WINAPI Playback(hWnd, wMsg, sSpeed, lpList)
|
| DESCRIPTION Journal Playback Function
|
| PARAMETERS  HWND hWnd       - Specifies handle to the window
|                                 to send notification to.
|       UINT wMsg       - Specifies notification messasge.
|       short sSpeed    - Specifies speed of playback.
|       LPRECORD lpList - Specifies pointer to the events list.
|
| RETURN     None.
|
*/
void WINAPI Playback(HWND hWnd, UINT wMsg, short sSpeed, LPRECORD lpList)
{

        if (bJournalBusy)
                return;

        if (lpList == NULL)
                return;

        hWndNotify = hWnd;
        wMsgNotify = wMsg;
        bJournalBusy = TRUE;
```

```c
        lpJmlList = lpList;
        sPlaybackSpeed = sSpeed;


        dwInitPlaybackTime = GetTickCount();
        dwPrevMsgTime = dwInitPlaybackTime;

        hJournalHook = SetWindowsHookEx(WH_JOURNALPLAYBACK, (FARPROC)PlayProc,
                                                    hInst, NULL);
        return;
}


/*-----------------------------------------------------------------------
|
| FUNCTION    void WINAPI HookFreeJournal(void)
|
| DESCRIPTION Release journal hook.
|
| PARAMETERS  None.
|
| RETURN     None.
|
*/
void WINAPI HookFreeJournal(void)
{

        if (hJournalHook)
        {
                UnhookWindowsHookEx(hJournalHook);
                bJournalBusy = FALSE;
                hJournalHook = NULL;
        }
}

/*-----------------------------------------------------------------------
|
| FUNCTION    static void RecNotify(void)
|
| DESCRIPTION Notify about end of recording.
|
| PARAMETERS  None.
|
| RETURN     None.
|
*/
static void RecNotify(void)
{
        LPRECORD lpList;
        DWORD   dwFirstTime;

        // reset the time field in all of these
        if (lpJmlList)
                dwFirstTime = lpJmlList->msg.time;

        lpList = lpJmlList;
```

```
        while (lpList != NULL)
        {
                lpList->msg.time -= dwFirstTime;
                lpList = lpList->pNext;
        }

        SendMessage(hWndNotify, wMsgNotify, 0, (LONG)lpJmlList);

}


/*-------------------------------------------------------------------
|
|
| FUNCTION    DWORD CALLBACK RecProc(nCode, wParam, IParam)
|
| DESCRIPTION The RecProc function is a callback function that records
|             messages that the system removes from the system message queue.
|
| PARAMETERS  int nCode      - Specifies whether the callback function
|                              should process the message or call the
|                              CallNextHookEx function. If this parameter
|                              is less than zero, the callback function
|                              should pass the message to CallNextHookEx
|                              without further processing.
|            WORD wParam     - Specifies a NULL value.
|            LONG IParam     - Points to an EVENTMSG structure that
|                              represents the message being processed
|                              by the callback function.
|
| RETURN      The callback function should return zero.
|
*/
DWORD CALLBACK RecProc(int nCode, WORD wParam, LONG IParam)
{
        static LPRECORD lpPrevList;  // Handle to prev recorded event
        static WORD    wNumEvents;       // ** number of events recorded ** for testing **
        static BOOL    bPause = FALSE;
        LPRECORD        lpList;
        LPEVENTMSG      lpEvent;
        BOOL            bCallNext = TRUE;
        DWORD           dwRetcode = 0;
        DWORD           dwTime;


        switch (nCode)
        {
                case HC_ACTION :
                        if (bPause)
                        {
                                break;
                        }
                        dwTime = GetTickCount();
                        lpEvent = (LPEVENTMSG) IParam;
                        if (lpEvent->message == WM_KEYDOWN && LOBYTE(lpEvent-
>paramL) == wStopKey)
                                {
```

```
                                        HookFreeJournal();
                                        RecNotify();
                                        break;
                        }
                        if (lpEvent->message >= WM_MOUSEFIRST && lpEvent->message <=
WM_MOUSELAST)
                        {
                                if (wMouRec == REC_MOUIGNORE)
                                {
                                        break;
                                }
                                else if (wMouRec == REC_MOUCLICK &&         lpEvent-
>message == WM_MOUSEMOVE)
                                {
                                        break;
                                }
                        }
                        // Allocate the next member (zeroinit it so hNext field doesn't
                        // have to be explicitly set to zero)
                        lpList = Gmalloc((DWORD) sizeof(RECORD));

                        if (lpList == NULL)
                        {
                                HookFreeJournal();
                                RecNotify();
                                break;
                        }

                        // Update the previous member to point to this new one.
                        if (lpJrnlList == NULL)
                        { // It's the first one
                                wNumEvents = 0;
                                lpJrnlList = lpList;
                        }
                        else
                        {
                                lpPrevList->pNext = lpList;
                        }
                        lpPrevList = lpList;

                        // Store the message in the new one

                        lpList->msg = *lpEvent;
                        lpList->msg.time = dwTime;
                        break;

                case HC_SYSMODALON:
                        bPause = TRUE;
                        break;

                case HC_SYSMODALOFF:
                        bCallNext = FALSE;
                        bPause = FALSE;
                        HookFreeJournal();
                        RecNotify();
                        break;
```

```
                · default :
                        break;
                }

        if (bCallNext)  {
                dwRetcode = CallNextHookEx(hJournalHook, nCode, wParam, lParam);
        }
        return dwRetcode;
}
```

```
/*---------------------------------------------------------------------
|
| FUNCTION    void WINAPI Record(hWnd, wMsg, wKey, wMou)
|
| DESCRIPTION Journal Record Function
|
| PARAMETERS  HWND hWnd        - Specifies handle to the window
|                        to send notification to.
|             UINT wMsg        - Specifies notification messasge.
|             UINT wKey        - Specifies stop key VK_ value.
|             UINT wMou        - Specifies type of mouse events that
|                        should be recorded.
|
| RETURN      None.
|
*/
void WINAPI Record(HWND hWnd, UINT wMsg, UINT wKey, UINT wMou)
{

        if (bJournalBusy)
                return;

        hWndNotify = hWnd;
        wMsgNotify = wMsg;
        wStopKey  = wKey;
        wMouRec   = wMou;
        lpJmlList = NULL;

        hJournalHook = SetWindowsHookEx(WH_JOURNALRECORD, (FARPROC)RecProc,
hInst, NULL);
        if (hJournalHook)
                bJournalBusy = TRUE;

}
```

```
                                 */
                                else   .
                                {
                                        switch (iScrlCom & SCRLS_ACT)
                                        {
                                                case SB_LINEUP:
                                                        /* line up
                                                        */
                                                        idWord = IDW_LINEUP;
                                                        break;
                                                case SB_LINEDOWN:
                                                        /* line down
                                                        */
                                                        idWord = IDW_LINEDOWN;
                                                        break;
                                                case SB_PAGEUP:
                                                        /* page up
                                                        */
                                                        idWord = IDW_PAGEUP;
                                                        break;
                                                case SB_PAGEDOWN:
                                                        /* page down
                                                        */
                                                        idWord = IDW_PAGEDOWN;
                                                        break;
                                        }
                                }
                                /* MDI frame is a spesial case
                                */
                                if (iScrlCom & SCRLS_MDI )
                                {
                                        len = wsprintf(
                                                szCaptionBuf, "%s %s",
                                                (LPSTR)UserGetDefWord(IDW_MDIFRAME),
(LPSTR)UserGetDefWord(idWord));
                                }
                                else
                                {
                                        lstrcpy(szCaptionBuf, UserGetDefWord(idWord));
                                        len = lstrlen(szCaptionBuf);
                                }
                                return(len);

}

#ifdef DEBUG_DLG

/*-------------------------------------------------------------------
|
| FUNCTION    _LOCAL int ContextPakWindDebug(hwnd)
|
| DESCRIPTION Get debug information for the given window.
|
| PARAMETERS  HWND hwnd - Specifies handle to the window we are looking at.
|
| RETURN     Length of the caption text.
```

```c
|
*/
_LOCAL int ContextPakWindDebug(HWND hwnd)
{
        /* Now we can recieve text from EDIT
        */
        return((int) SendMessage(hwnd. WM_GETTEXT. sizeof(szCaptionBuf) - 1 .
(LONG)(LPSTR)szCaptionBuf));
}


/*--------------------------------------------------------------------
|
| FUNCTION    _LOCAL int ContextPakDebug(void)
|
| DESCRIPTION Create debug string.
|
| PARAMETERS  None.
|
| RETURN      Length of the caption text.
|
*/
_LOCAL int ContextPakDebug(void)
{
        /* ADD DEBUG INFO TO THE CONTEXT STRING
        */
        HWND hwnd = pciLast->hwnd;
        PSTR Str;
        int len = lstrlen(szCaptionBuf);
        int lend;

        if (! len)
        {
                switch (pciLast->conType)
                {
                        case CON_WIND:
                        case CON_ICON:
                                /* Add window debuf info
                                */
                                len = ContextPakWindDebug(hwnd);
                                break;

                        default:
                                :

                }

                if (! len)
                {
                        /* No text for this item
                        */
                        lstrcpy(szCaptionBuf, "<No Caption>");
                        len = lstrlen(szCaptionBuf);
                }

        }
```

```c
/* Move start pointer
*/
Str  = szCaptionBuf + len;

/* Show the handle and the parent handle for the related window.
*/
lend = wsprintf(Str, "\t%1d %04X\t", pciLast->iLevel, hwnd);
Str  += lend;

/* Add debug info to the string.
*/
switch (pciLast->conType)
{

        case CON_WIND:
        case CON_ICON:
                /* Its a window or a control.
                */
                if (! hwnd)
                        /* No associated window ?
                        */
                        break;

                /* Parent and owner
                */
                lend = wsprintf(Str, "%04X %04X ", GetParent(hwnd), GetWindow(hwnd,
GW_OWNER));

                /* Add the class name to it.
                */
                GetClassName(hwnd, Str+lend, MAXSTRING);

                /* Usefull properties
                */
                if (! IsWindowEnabled(hwnd))
                        lstrcat(Str, " <INACTIVE>");
                else if (! IsWindowVisible(hwnd))
                        lstrcat(Str, " <INVISIBLE>");
                else if (IsZoomed(hwnd))
                        lstrcat(Str, " <MAXIMIZED>");
                else if (IsIconic(hwnd))
                        lstrcat(Str, " <MINIMIZED>");
                if (hwnd == GetActiveWindow())
                        lstrcat(Str, " <ACTIVE>");
                if (hwnd == GetFocus())
                        lstrcat(Str, " ,<FOCUS>");

                /* We need to return this
                */
                lend = lstrlen(Str);
                break;

        case CON_SYSCOM:
                /* System commans
                */
                lend = wsprintf(Str, "<SYSTEM COMMAND %d>", pciLast->u.SysCom);
```

```
                              break;

                case CON_MENUPOPUP:
                        /* Popup menu properties
                        */
                        lend = wsprintf(Str, "%04x <POPUP MENU %d>",
                                GetMenuState(pciLast->u.MenuPop.hMenu, pciLast-
>u.MenuPop.iEntry,

                                MF_BYPOSITION), pciLast->u.MenuPop.iEntry);
                        break;

                case CON_MENU:
                        /* Menu item properties
                        */
                        lend = wsprintf(Str, "<MENU ITEM %d>", pciLast->u.Menu.id);
                        break;

                case CON_ACCEL:
                        /* Accelerator
                        */
                        lend = wsprintf(Str, "<ACCELERATOR FOR %d>", pciLast->u.Acc.id);
                        break;

                case CON_LAUNCH:
                        /* ProgMan launch command
                        */
                        lend = wsprintf(Str, "<%s>", (LPSTR)(pciLast->u.PMItem.szFile));
                        break;

                case CON_MACRO:
                        /* Macro
                        */
                        lend = wsprintf(Str, "<MACRO>");
                        break;
        }

        /* Calculate maximum length
        */
        if (lend > iDebugCapLen)
                iDebugCapLen = lend;

        return(len);
}

#endif

/*-------------------------------------------------------------------
|
| FUNCTION    _LOCAL int ContextPak(void)
|
| DESCRIPTION Build a context string for the context block.
|         User pciLast to identify the object.
|
| PARAMETERS  None.
|
| RETURN      Length of the caption text.
```

```
/*
** File: PLAYBACK.C
**
** Functions for Macro Execution
**
** Public functions:   MakeHookReady
**              VCM_Execute
**
** Private Functions : me_SingleCommand
**              me_Clk
**              me_Key
**              me_String
**              me_Execute
**
**********************************************************************
**********************************************************************/


#define WIN31          // need this to use extended 3.1 functionality
#include <windows.h>

#include <shellapi.h>
#include <ctype.h>

#include "vtools.h"
#include "vc.h"

/* Private Function Prototypes
*/
_LOCAL BOOL me_SingleCommand(LPMACRO, HWND);
_LOCAL BOOL me_Clk(LPMACRO);
_LOCAL BOOL me_Key(VCM_KEY KeyType);
_LOCAL BOOL me_String(LPSTR Str);
_LOCAL BOOL me_Execute(LPSTR Str);


/*-------------------------------------------------------------------
|
| FUNCTION    BOOL MakeHookReady(void)
|
| DESCRIPTION Wait until we finish playback.
|
| PARAMETERS  None.
|
| RETURN      TRUE if success.
|
*/
BOOL MakeHookReady(void)
{
        MSG msg;

        while (HookJournalBusy())
        {
                if (PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE))
                        ProcessMessage(msg);
        }
```

```
        return TRUE;
}

/*-----------------------------------------------------------------------
|
| FUNCTION    BOOL VCM_Execute(LPMACRO CmdPtr, HWND hGlobalWnd)
|
| DESCRIPTION Processes the command encoded in the input
|        command struture.
|
| PARAMETERS  LPMACRO CmdPtr    - Points to an list of MACRO elements.
|        HWND    hGlobalWnd - Default window to send commands to.
|
| RETURN      TRUE if success.
|
*/
BOOL VCM_Execute(LPMACRO CmdPtr, HWND hGlobalWnd)
{
        WORD wErr;
        HWND hLocalWnd;

        /* Check for NULL pointers
        */
        if (CmdPtr == NULL)
                return 1;

        while (CmdPtr != NULL)
        {
                /* use currently active win
                */
                if ((CmdPtr->cmdType == CMD_KEY) ||
                        (CmdPtr->cmdType == CMD_TEXT) ||
                        (CmdPtr->cmdType == CMD_LAUNCH))
                        hLocalWnd = NULL;
                else
                        hLocalWnd = hGlobalWnd;

                /* Process a single command
                */
                if (wErr = me_SingleCommand(CmdPtr, hLocalWnd))
                        return wErr;

                /* Get the next command
                */
                CmdPtr = CmdPtr->pNext;
        }
        return TRUE;
}


/*-----------------------------------------------------------------------
|
| FUNCTION    _LOCAL BOOL me_SingleCommand(LPMACRO CmdPtr, HWND hWnd)
|
| DESCRIPTION Execute single macro command.
|
```

```
|  PARAMETERS  LPMACRO CmdPtr    - Points to an list of MACRO elements.
|         HWND   hGlobalWnd - Default window to send commands to.
|
|  RETURN    TRUE if success.
|
*/
_LOCAL BOOL  me_SingleCommand(LPMACRO CmdPtr, HWND hWnd)
{
        RECT    rect;
        POINT   pt;

        BOOL    bFoundIt;
        HMENU   hMenu;
        WORD    wTotal, wFlags, i, KeyPos;
        MACRO macro;
        WORD    wKeyUp, wKeyDown;
        int     iLevel;

        /* Was a specific window given or are we to assume that we should use
        ** the currently active window?
        */
        if (! hWnd)
                hWnd = GetActiveWindow();

        /* Make sure it is a valid window handle
        */
        if (! IsWindow(hWnd))
                return FALSE;

        /* Was a class specified and if so was there also window text given.
        ** Don't allow specification of window text without the window
        ** class being given as well.
        */

        /* Determine the type of command and process the command specific action.
        */
        switch (CmdPtr->cmdType)
        {
                case CMD_MENU :
                        /* Verify that the given window has a menu (do I need to bother w/this?)
                        */
                        if (!(hMenu = GetMenu(hWnd)))
                                return FALSE;
                        /* Check to make sure selection is available
                        */
                        i = GetMenuState(hMenu, CmdPtr->Cmd.Menu.id, MF_BYCOMMAND);
                        if ((i & MF_DISABLED) || (i & MF_GRAYED) || (i == -1))
                                break;

                        /* Clear the menus before the command is sent
                        */
                        iLevel = HookGet_MenuLevel();
                        while (iLevel-- >= 0)
                        {
                                PostMessage(hWnd, WM_SYSKEYDOWN, VK_ESCAPE, 0L);
                                Yield();
```

```c
                        }

                        PostMessage(hWnd, WM_COMMAND, CmdPtr->Cmd.Menu.id, 0L);
                        break;

                case CMD_MENUPOPUP :
                        /* Verify that the given window has a menu (do I need to bother w/this?)
                        */
                        if (!(hMenu = GetMenu(hWnd)))
                                return FALSE;
                        /* Test to see where the current menu hilighting is.
                        */
                        hMenu = HookGet_MenuAtLevel(0);

                        /* No menu up - Activate the Menu Bar
                        */
                        if (!hMenu)
                        {
                                hMenu = GetMenu(hWnd);
                                PostMessage(hWnd, WM_SYSCOMMAND, SC_KEYMENU,
0L);

                                i = CmdPtr->Cmd.MenuPopup.iKeyPos;
                                while (i--)
                                {
                                        PostMessage(hWnd, WM_SYSKEYDOWN, VK_RIGHT,
0L);

                                        Yield();
                                }

                                /* Need to check to see if there really is a menu to pop up or
                                ** if it is a menu item on the menu bar that has no pulldown.
                                */
                                if ((i = GetMenuItemID(hMenu, CmdPtr-
>Cmd.MenuPopup.iKeyPos)) != -1)
                                {
                                        iLevel = HookGet_MenuLevel();
                                        while (iLevel-- >= 0)
                                        {
                                                PostMessage(hWnd, WM_SYSKEYDOWN,
VK_ESCAPE, 0L);

                                                Yield();
                                        }

                                        PostMessage(hWnd, WM_COMMAND, i, 0L);
                                }
                                else
                                        PostMessage(hWnd, WM_SYSKEYDOWN,
VK_DOWN, 0L);
                        }
                        /* It's a cascading popup
                        */
                        else
                        {

                                /* Pop "back" the menus to the correct level
                                */
```

```
                                        while(HookGet_Menu(CmdPtr->Cmd.MenuPopup.wLevel + 1))
                                        {
                                                PostMessage(hWnd, WM_SYSKEYDOWN,

VK_ESCAPE, 0L);                                 Yield();
                                        }

                                        /* Get the current position that is hilighted
                                        */
                                        hMenu = HookGet_MenuAtLevel(CmdPtr-
>Cmd.MenuPopup.wLevel);

                                        wTotal = GetMenuItemCount(hMenu);

                                        i = 0;
                                        KeyPos = 0;
                                        bFoundIt = FALSE;
                                        while ((i < wTotal) && !bFoundIt)
                                        {
                                                wFlags = GetMenuState(hMenu, i, MF_BYPOSITION);
                                                if (wFlags & MF_HILITE)
                                                        bFoundIt = TRUE;
                                                else
                                                {
                                                        if ((wFlags & MF_POPUP) || (!(wFlags &
MF_SEPARATOR)))

                                                                KeyPos++;
                                                        i++;
                                                }
                                        }

                                        /* Must take separators into account in position
                                        */
                                        i = KeyPos;

                                        if (CmdPtr->Cmd.MenuPopup.wLevel)
                                        {
                                                wKeyUp          = VK_UP;
                                                wKeyDown = VK_DOWN;
                                        }
                                        else
                                        {
                                                wKeyUp          = VK_LEFT;
                                                wKeyDown = VK_RIGHT;
                                        }

                                        if (i < (WORD)CmdPtr->Cmd.MenuPopup.iKeyPos)
                                        {
                                                i = CmdPtr->Cmd.MenuPopup.iKeyPos - i;
                                                while (i--)
                                                {
                                                        PostMessage(hWnd, WM_SYSKEYDOWN,
wKeyDown, 0L);

                                                }
                                        }
                                        else
                                        {
```

```c
                                                if (i > (WORD)CmdPtr->Cmd.MenuPopup.iKeyPos)
                                                {
                                                        i = i - CmdPtr->Cmd.MenuPopup.iKeyPos;
                                                        while (i--)
                                                        {
                                                                PostMessage(hWnd,
    WM_SYSKEYDOWN, wKeyUp, 0L);
                                                        }
                                                }

                                        PostMessage(hWnd, WM_SYSKEYDOWN, VK_RETURN, 0L);
                                }

                        break;

                case CMD_SYSTEM :

                                if ((CmdPtr->Cmd.System.wCmd == SC_KEYMENU) || (CmdPtr-
    >Cmd.System.wCmd == SC_MOUSEMENU))
                                {
                                        /* Activating the system menu of an iconized window can't be
    done
                                        ** with the normal syscommands and syskeys.
                                        ** Using mouse commands works but it has the unpleasant side
    effect
                                        ** of moving the pointer.  Therefore this may not be an
    acceptable
                                        ** solution.
                                        */
                                        if (GetParent(hWnd))
                                        {
                                                /* This combination seems to work in all cases except
    for activating
                                                ** the system menu of a child window in Excel that is
    not maximized.
                                                */
                                                PostMessage(hWnd, WM_SYSCOMMAND, CmdPtr-
    >Cmd.System.wCmd, 0L);
                                                PostMessage(hWnd, WM_SYSKEYDOWN,
    VK_RETURN, 0L);
                                        }
                                        else
                                        {
                                                PostMessage(hWnd, WM_SYSCOMMAND,
    SC_KEYMENU, 0L);
                                                PostMessage(hWnd, WM_SYSKEYDOWN,
    VK_SPACE, 0L);
                                        }
                                }
                                else
                                {
                                        iLevel = HookGet_MenuLevel();
                                        while (iLevel-- >= 0)
                                        {
```

```
                                          PostMessage(hWnd, WM_SYSKEYDOWN,
VK_ESCAPE, 0L);
                                          Yield();
                              }

                              PostMessage(hWnd, WM_SYSCOMMAND, CmdPtr-
>Cmd.System.wCmd, 0L);
                          }

                  break;

          case CMD_MESSAGE :
                  /* Just message to post
                  */
                  PostMessage(hWnd, CmdPtr->Cmd.Msg.wMsg, CmdPtr-
>Cmd.Msg.wParam,   CmdPtr->Cmd.Msg.lParam);
                  break;

          case CMD_SELECT :
          {
                  /* Bring hWnd to the top and activate it.
                  */
                  POINT pt;
                  int i;

                  if (GetWindowLong(hWnd, GWL_STYLE) & WS_CHILD)
                  {
                          SetFocus(hWnd);
                          GetWindowRect(hWnd, &rect);

                          pt.x = rect.left;
                          pt.y = rect.top;
                          for (i = 0; i < 5; i ++)
                          {
                                  if (WindowFromPoint(pt) == hWnd)
                                          break;
                                  pt.x ++;
                                  pt.y ++;
                          }
                          macro.cmdType  = CMD_MOUSE;
                          macro.pNext    = NULL;
                          macro.szWndClass = NULL;
                          macro.szDesc   = NULL;
                          macro.Cmd.Mouse.mouType   = MOU_LBCLK;
                          macro.Cmd.Mouse.bPosType = VCM_MP_SCREEN;
                          macro.Cmd.Mouse.wX     = pt.x;
                          macro.Cmd.Mouse.wY     = pt.y;
                          macro.Cmd.Mouse.CtrlPressed = 0;
                          macro.Cmd.Mouse.ShiftPressed = 0;
                          macro.Cmd.Mouse.AltPressed = 0;

                          VCM_Execute(&macro, hWnd);
                  }
                  else
                  {
                          BringWindowToTop(hWnd);
```

```c
                }
                break;
        }


        /* Mouse, Keyboard, and Journal Playback commands will all be handled via
        ** a Journal Playback Hook.  We still need to go through the window
        ** checking above to make sure that if the events are to go to a specific
        ** window that the window is there.
        */

        case CMD_MOUSE :

                /* For all mouse commands, convert any client coordinates
                ** to screen coordinates before proceeding further.
                */
                if (CmdPtr->Cmd.Mouse.bPosType == VCM_MP_CLIENT)
                {
                        pt.x = CmdPtr->Cmd.Mouse.wX;
                        pt.y = CmdPtr->Cmd.Mouse.wY;
                        ClientToScreen(hWnd, (LPPOINT) &pt);
                        CmdPtr->Cmd.Mouse.wX = pt.x;
                        CmdPtr->Cmd.Mouse.wY = pt.y;
                        /* in case it's used later
                        */
                        CmdPtr->Cmd.Mouse.bPosType = VCM_MP_SCREEN;
                }


                switch (CmdPtr->Cmd.Mouse.mouType)
                {
                        case MOU_MOVE :
                                /* Do moves need to be done via playback or is this
OK???

                                */
                                SetCursorPos(CmdPtr->Cmd.Mouse.wX, CmdPtr->Cmd.Mouse.wY);

                                break;

                        /* Is it necessary to set the focus for clicks and double clicks?
                        */
                        case MOU_LBDBLCLK :   // Double Clicks
                        case MOU_RBDBLCLK :
                        case MOU_MBDBLCLK :
                        case MOU_LBCLK :      // Single Clicks
                        case MOU_RBCLK :
                        case MOU_MBCLK :
                                return (me_Clk(CmdPtr));
                                break;
                }
                break;


        case CMD_KEY :
                /* May need more values passed in for the OEM scan code to be set.
                ** Is it necessary to set the focus here before the key is sent?
```

```c
                            ** if window is inconized or ALT is pressed then WM_SYSKEY
                            */
                            return (me_Key(CmdPtr->Cmd.Key));

                            break;

            case CMD_TEXT :

                            return (me_String(CmdPtr->szDesc));

                            break;

            case CMD_LAUNCH :

                            return (me_Execute(CmdPtr->szDesc));

                            break;


            case CMD_JOURNAL :
            {
                    LPRECORD pFirstRecord;
                    LPRECORD pRecord;
                    POINT pt;

                    if (HookJournalBusy())
                            return FALSE;

                    /* need to define how the playback list is going to be sent and what
                    ** we are going to do about any timing type problems such as windows
                    ** taking longer to appear than they did in the original recording etc.
                    */
                    pFirstRecord = RecordMake(CmdPtr->Cmd.Journal.pRecord);
                    for (pRecord = pFirstRecord; pRecord != NULL; pRecord = pRecord->pNext)
                    {
                            if (pRecord->msg.message >= WM_MOUSEFIRST &&
pRecord->msg.message <= WM_MOUSELAST)
                            {
                                    pt.x = pRecord->msg.paramL;
                                    pt.y = pRecord->msg.paramH;
                                    ClientToScreen(hWnd, &pt);
                                    pRecord->msg.paramL = pt.x;
                                    pRecord->msg.paramH = pt.y;
                            }
                    }
                    Playback(NULL, 0, 0, pFirstRecord);
                    break;
            }

            default :
                    /* error - Unknown Command Type
                    */
                    return FALSE;
                    break;
    }
```

```c
        return TRUE;
}


#define MAKEKEY(uVKey) (MAKEWORD(uVKey, MapVirtualKey(uVKey, 0)))

/*------------------------------------------------------------------------
|
| FUNCTION   _LOCAL BOOL me_Clk(LPMACRO CmdPtr)
|
| DESCRIPTION Execute mouse macro command.
|
| PARAMETERS  LPMACRO CmdPtr   - Points to an list of MACRO elements.
|
| RETURN    TRUE if success.
|
*/
_LOCAL BOOL me_Clk(LPMACRO CmdPtr)
{
        LPRECORD lpList, lpHead;
        WORD    Down, DownSec, Up;
        WORD    time = 0x50;
        BOOL    bSysKey =   (CmdPtr->Cmd.Mouse.AltPressed) && ! (CmdPtr-
>Cmd.Mouse.CtrlPressed);
        POINT   ptCur;

        GetCursorPos(&ptCur);

        /* Mouse coordinates have already been converted to screen coordinates
        */
        switch (CmdPtr->Cmd.Mouse.mouType)
        {
                case MOU_LBCLK :
                        Down = WM_LBUTTONDOWN;
                        DownSec  = NULL;
                        Up  = WM_LBUTTONUP;
                        break;
                case MOU_RBCLK :
                        Down = WM_RBUTTONDOWN;
                        DownSec  = NULL;
                        Up  = WM_RBUTTONUP;
                        break;
                case MOU_MBCLK :
                        Down = WM_MBUTTONDOWN;
                        DownSec  = NULL;
                        Up  = WM_MBUTTONUP;
                        break;
                case MOU_LBDBLCLK :
                        Down = WM_LBUTTONDOWN;
                        DownSec  = WM_LBUTTONDBLCLK;
                        Up  = WM_LBUTTONUP;
                        break;
                case MOU_RBDBLCLK :
                        Down = WM_RBUTTONDOWN;
                        DownSec  = WM_RBUTTONDBLCLK;
```

```
                        Up  = WM_RBUTTONUP;
                        break;
                case MOU_MBDBLCLK :
                        Down = WM_MBUTTONDOWN;
                        DownSec  = WM_MBUTTONDBLCLK;
                        Up  = WM_MBUTTONUP;
                        break;
                default:
                        return FALSE;

        }


        lpList = Gmalloc((DWORD) sizeof(RECORD));
        lpHead = lpList;

        if (lpList)
        {
                lpList->msg.message = WM_MOUSEMOVE;
                lpList->msg.paramL = CmdPtr->Cmd.Mouse.wX;
                lpList->msg.paramH  = CmdPtr->Cmd.Mouse.wY;
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
                return FALSE;

        if (CmdPtr->Cmd.Mouse.AltPressed)
        {
                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));
                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = WM_SYSKEYDOWN;
                        lpList->msg.paramL  = MAKEKEY(VK_MENU);
                        lpList->msg.paramH  = 0x1;   // repeat count
                        lpList->msg.time    = time;
                        time += 0x50;
                }
                else
                        return FALSE;
        }

        if (CmdPtr->Cmd.Mouse.CtrlPressed)
        {
                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = WM_KEYDOWN;
                        lpList->msg.paramL  = MAKEKEY(VK_CONTROL);
                        lpList->msg.paramH  = 0x1;   // repeat count
                        lpList->msg.time    = time;
                        time += 0x50;
                }
                else
```

```
                                return FALSE;
                }

                if (CmdPtr->Cmd.Mouse.ShiftPressed)
                {
                        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                        if (lpList->pNext)
                        {
                                lpList = lpList->pNext;
                                lpList->msg.message = bSysKey ? WM_SYSKEYDOWN :
WM_KEYDOWN;
                                lpList->msg.paramL = MAKEKEY(VK_SHIFT);
                                lpList->msg.paramH = 0x1;   // repeat count
                                lpList->msg.time   = time;
                                time += 0x50;
                        }
                        else
                                return FALSE;
                }

                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = Down;
                        lpList->msg.paramL = CmdPtr->Cmd.Mouse.wX;
                        lpList->msg.paramH = CmdPtr->Cmd.Mouse.wY;
                        lpList->msg.time   = time;
                        time += 0x50;
                }
                else
                        return FALSE;

                if (DownSec)
                {
                        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                        if (lpList->pNext)
                        {
                                lpList = lpList->pNext;
                                lpList->msg.message = Up;
                                lpList->msg.paramL = CmdPtr->Cmd.Mouse.wX;
                                lpList->msg.paramH = CmdPtr->Cmd.Mouse.wY;
                                lpList->msg.time   = time;
                                time += 0x50;
                        }
                        else
                                return FALSE;

                        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                        if (lpList->pNext)
                        {
```

```
                      lpList = lpList->pNext;
                      lpList->msg.message = DownSec;
                      lpList->msg.paramL  = CmdPtr->Cmd.Mouse.wX;
                      lpList->msg.paramH  = CmdPtr->Cmd.Mouse.wY;
                      lpList->msg.time    = time;
                      time += 0x50;
              }
              else
                      return FALSE;


      }
      lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

      if (lpList->pNext)
      {
              lpList = lpList->pNext;
              lpList->msg.message = Up;
              lpList->msg.paramL  = CmdPtr->Cmd.Mouse.wX;
              lpList->msg.paramH  = CmdPtr->Cmd.Mouse.wY;
              lpList->msg.time    = time;
              time += 0x50;
      }
      else
              return FALSE;

      if (CmdPtr->Cmd.Mouse.ShiftPressed)
      {
              lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

              if (lpList->pNext)
              {
                      lpList = lpList->pNext;
                      lpList->msg.message = bSysKey ? WM_SYSKEYUP : WM_KEYUP;
                      lpList->msg.paramL  = MAKEKEY(VK_SHIFT);
                      lpList->msg.paramH  = 0x1;    // repeat count
                      lpList->msg.time    = time;
                      time += 0x50;
              }
              else
                      return FALSE;
      }

      if (CmdPtr->Cmd.Mouse.CtrlPressed)
      {
              lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

              if (lpList->pNext)
              {
                      lpList = lpList->pNext;
                      lpList->msg.message = WM_KEYUP;
                      lpList->msg.paramL  = MAKEKEY(VK_CONTROL);
                      lpList->msg.paramH  = 0x1;    // repeat count
                      lpList->msg.time    = time;
                      time += 0x50;
              }
              else
```

```c
                                return FALSE;
        }

        if (CmdPtr->Cmd.Mouse.AltPressed)
        {
                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = WM_KEYUP;
                        lpList->msg.paramL = MAKEKEY(VK_MENU);
                        lpList->msg.paramH = 0x1;   // repeat count
                        lpList->msg.time    = time;
                        time += 0x50;
                }
                else
                        return FALSE;
        }

        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

        if (lpList->pNext)
        {
                lpList = lpList->pNext;
                lpList->msg.message = WM_MOUSEMOVE;
                lpList->msg.paramL  = ptCur.x;
                lpList->msg.paramH  = ptCur.y;
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
                return FALSE;

        if (! MakeHookReady())
                return FALSE;
        else
                Playback(NULL, 0, -1, lpHead);

        return TRUE;

}

/*-------------------------------------------------------------------
|
| FUNCTION   _LOCAL BOOL me_Key(KeyType)
|
| DESCRIPTION Execute key macro command.
|
| PARAMETERS  VCM_KEY KeyType - Specifies ke description struct.
|
| RETURN     TRUE if success.
|
*/
_LOCAL BOOL me_Key(VCM_KEY KeyType)
{
```

```
LPRECORD lpList, lpHead;
WORD    time = 0x50;
BOOL    bSysKey = (KeyType.AltPressed) && ! (KeyType.CtrlPressed);
POINT   ptCur;

GetCursorPos(&ptCur);

/* Not quite sure why something like a mouse move must be sent
** before the key down to have the key down be recognized.
*/
lpList = Gmalloc((DWORD) sizeof(RECORD));

lpHead = lpList;

if (lpList)
{
        lpList->msg.message = WM_MOUSEMOVE;
        lpList->msg.paramL  = ptCur.x;
        lpList->msg.paramH  = ptCur.y;
        lpList->msg.time    = time;
        time += 0x50;
}
else
        return FALSE;

if (KeyType.AltPressed)
{
        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

        if (lpList->pNext)
        {
                lpList = lpList->pNext;
                lpList->msg.message = WM_SYSKEYDOWN;
                lpList->msg.paramL  = MAKEKEY(VK_MENU);
                lpList->msg.paramH  = 0x1;   // repeat count
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
                return FALSE;
}

if (KeyType.CtrlPressed)
{
        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

        if (lpList->pNext)
        {
                lpList = lpList->pNext;
                lpList->msg.message = WM_KEYDOWN;
                lpList->msg.paramL  = MAKEKEY(VK_CONTROL);
                lpList->msg.paramH  = 0x1;   // repeat count
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
```

```c
                        return FALSE;
        }

        if (KeyType.ShiftPressed)
        {
                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = bSysKey ? WM_SYSKEYDOWN :
WM_KEYDOWN;
                        lpList->msg.paramL  = MAKEKEY(VK_SHIFT);
                        lpList->msg.paramH  = 0x1;   // repeat count
                        lpList->msg.time    = time;
                        time += 0x50;
                }
                else
                        return FALSE;
        }

        if (KeyType.cKey)
        {

                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = bSysKey ? WM_SYSKEYDOWN :
WM_KEYDOWN;
                        lpList->msg.paramL  = MAKEKEY(KeyType.cKey);
                        lpList->msg.paramH  = 0x1;   // repeat count
                        lpList->msg.time    = time;
                        time += 0x50;
                }
                else
                        return FALSE;

                lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                if (lpList->pNext)
                {
                        lpList = lpList->pNext;
                        lpList->msg.message = bSysKey ? WM_SYSKEYUP : WM_KEYUP;
                        lpList->msg.paramL  = MAKEKEY(KeyType.cKey);
                        lpList->msg.paramH  = 0x1;   // repeat count
                        lpList->msg.time    = time;
                        time += 0x50;
                }
                else
                        return FALSE;
        }

        if (KeyType.ShiftPressed)
        {
```

```c
        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

        if (lpList->pNext)
        {
                lpList = lpList->pNext;
                lpList->msg.message = bSysKey ? WM_SYSKEYUP : WM_KEYUP;
                lpList->msg.paramL  = MAKEKEY(VK_SHIFT);
                lpList->msg.paramH = 0x1;    // repeat count
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
                return FALSE;
}

if (KeyType.CtrlPressed)
{
        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

        if (lpList->pNext)
        {
                lpList = lpList->pNext;
                lpList->msg.message = WM_KEYUP;
                lpList->msg.paramL  = MAKEKEY(VK_CONTROL);
                lpList->msg.paramH = 0x1;    // repeat count
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
                return FALSE;
}

if (KeyType.AltPressed)
{
        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

        if (lpList->pNext)
        {
                lpList = lpList->pNext;
                lpList->msg.message = (
                        ! (KeyType.cKey) ||
                        ! (KeyType.CtrlPressed) ||
                        ! (KeyType.ShiftPressed)
                ) ? WM_SYSKEYUP : WM_KEYUP;
                lpList->msg.paramL  = MAKEKEY(VK_MENU);
                lpList->msg.paramH = 0x1;    // repeat count
                lpList->msg.time    = time;
                time += 0x50;
        }
        else
                return FALSE;
}

if (! MakeHookReady())
        return FALSE;
else
```

```c
            _Playback(NULL, 0, -1, lpHead);

        return TRUE;
}

/*------------------------------------------------------------------
|
| FUNCTION    _LOCAL BOOL me_String(LPSTR Str)
|
| DESCRIPTION Execute string macro command.
|
| PARAMETERS  LPSTR Str - Specifies sourse string.
|
| RETURN      TRUE if success.
|
*/
_LOCAL BOOL me_String(LPSTR Str)
{
        LPRECORD lpList, lpHead;
        POINT   ptCur;
        LONG time=0x50;

        if (Str == NULL)
                return FALSE;

        GetCursorPos(&ptCur);

        /* Not quite sure why something like a mouse move must be sent
        ** before the key down to have the key down be recognized.
        */
        lpList = Gmalloc((DWORD) sizeof(RECORD));
        lpHead = lpList;

        if (lpList)
        {
                lpList->msg.message = WM_MOUSEMOVE;
                lpList->msg.paramL  = ptCur.x;
                lpList->msg.paramH  = ptCur.y;
                lpList->msg.time    = 0x50;
        }
        else
                return FALSE;


        while (*Str != NULL)
        {

                if (isupper(*Str))
                {
                        lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                        if (lpList->pNext)
                        {
                                lpList = lpList->pNext;
                                lpList->msg.message = WM_KEYDOWN;
                                lpList->msg.paramL  = MAKEKEY(VK_SHIFT);
```

```c
                                      lpList->msg.paramH  = 0x1;   // repeat count
                                      lpList->msg.time    = time+=0x20;
                     }
                     else

                            return FALSE;
            }


            lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

            if (lpList->pNext)
            {
                     lpList = lpList->pNext;
                     lpList->msg.message = WM_KEYDOWN;
                     lpList->msg.paramL  = MAKEKEY(toupper(*Str));
                     lpList->msg.paramH  = 0x1;   // repeat count
                     lpList->msg.time    = time+=0x20;
            }
            else
                     return FALSE;

            lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

            if (lpList->pNext)
            {
                     lpList = lpList->pNext;
                     lpList->msg.message = WM_KEYUP;
                     lpList->msg.paramL  = MAKEKEY(toupper(*Str));
                     lpList->msg.paramH  = 0x1;   // repeat count
                     lpList->msg.time    = time+=0x20;
            }
            else
                     return FALSE;

            if (isupper(*Str))
            {
                     lpList->pNext = Gmalloc((DWORD) sizeof(RECORD));

                     if (lpList->pNext)
                     {
                            lpList = lpList->pNext;
                            lpList->msg.message = WM_KEYUP;
                            lpList->msg.paramL  = MAKEKEY(VK_SHIFT);
                            lpList->msg.paramH  = 0x1;   // repeat count
                            lpList->msg.time    = time+=0x20;
                     }
                     else
                            return FALSE;
            }

            Str++;
    }

    if (! MakeHookReady())
            return FALSE;
    else
```

```
                 Playback(NULL, 0, -1, lpHead);

         return TRUE;
}


/*--------------------------------------------------------------------
|
| FUNCTION    _LOCAL BOOL me_Execute(LPSTR Str)
|
| DESCRIPTION Execute launch macro command.
|
| PARAMETERS  LPSTR Str - Specifies command string
|
| RETURN     TRUE if success.
|
*/
_LOCAL BOOL me_Execute(LPSTR Str)
{
        char szExec[MAXFILENAME + 1];
        char * pszParam;

        lstrcpy(szExec, Str);
        for (pszParam = szExec; *pszParam != '\0' ; pszParam ++)
        {
                if (*pszParam == ' ')
                {
                        *pszParam = '\0';
                        pszParam ++;
                        break;
                }
        }
        if (ShellExecute(NULL, NULL, (LPSTR)szExec, (LPSTR)pszParam, NULL,
SW_SHOWNORMAL) < 32)
        {
                Error(ERRAppExec, (LPSTR)Str);
                return FALSE;
        }

        return TRUE;
}
```

```
/*
** File: STATUS.C
**
** This is the windows display interface for the status window.
**
** Public functions:    StatusSetPref
**                      PhraseListAdd
**                      StatusInit
**                      StatusCheckMsg
**                      StatusGetWindow
**
** Exported functions:  PhraseTimerProc
**                      StatusWndProc
**
** Private functions:   StatusBarPer
**                      StatusBarDraw
**                      StatusBars
**                      StatusChange
**                      PhraseFind
**                      CloseCallFind
**                      PhraseListMove
**                      PhraseListInc
**                      PhraseListSetup
**                      PhraseDrawItem
**                      PhraseExec
**                      StartTimer
**                      StopTimer
**                      SelectOurFont
**
***************************************************************************
***************************************************************************/

#define WIN31                // need this to use extended 3.1 functionality
#include <windows.h>

#include <memory.h>
#include <stdlib.h>

#include "vtools.h"

#include "vc.h"
#include "vcrc.h"            /* only files included by vc.rc */
#include "vchelp.h"          /* only file included by vchlp.hpj */

#define PROMPT_LEN 14
#define IDT_PHRASE 1
#define IDLIST_PHRASE    4
#define BMP_SIZE        16

/*------------------------------------------------------------------
|
| Menu
|
*/
enum
{
```

```
                IDM_PREFS = MENU_STATUS,
                IDM_TRAIN,
                IDM_EDIT,
                IDM_PAUSE,
                IDM_EXIT,
                IDM_HELPCONTENT,
                IDM_HELPSEARCH,
                IDM_HELPONHELP,
                IDM_ABOUT
};


/*-----------------------------------------------------------------------
|
| Strings
|
*/
enum
{
                IDS_TITLE = IDS_STATUS,
                IDS_DEBUG,
                IDS_PAUSE,
                IDS_CONFID,
                IDS_VOLUME,
                IDS_NEW,
                IDS_QUERY
};

/*-----------------------------------------------------------------------
|
| System menu additions.  NOTE: leave low 4 bits unused !
|
*/
#define IDM_SYSDEBUG       (0x0110)



/*-----------------------------------------------------------------------
|
| Communucation with Editor
|
*/
_LOCAL char szFrameClass[]  = "VoiceEditFrame";
_LOCAL UINT iEditChangeMsg = NULL;

_LOCAL char szStatusClass[] = "VoiceStatus";

_LOCAL HWND hwndStatus = NULL;
_LOCAL HWND hwndList = NULL;

_LOCAL HANDLE hAccTableStatus;

_LOCAL WORD PhraseTimer = NULL;

_LOCAL int  iVolumeMin = 20;
_LOCAL int  iVolumeMax = 80;

_LOCAL UINT wCloseCallInc = 0;
```

```
_LOCAL BOOL bCloseCallWas = FALSE;
_LOCAL UINT wCloseCallNumber;
_LOCAL UINT wUttCloseCall = 0;

_LOCAL int iStatusSizeMin;
_LOCAL BOOL bPause = FALSE;

_LOCAL HICON   hicoMain;
_LOCAL HICON   hicoStat;

_LOCAL HBITMAP hbmpPaint;
_LOCAL HBITMAP hbmpAnd;

_LOCAL HFONT  hFontCur = NULL;
_LOCAL int    cxStatusText;
_LOCAL int    cyStatusText;

_LOCAL RECOGRES vrState;

/*-------------------------------------------------------------------------
|
| FUNCTION   _LOCAL int StatusBarPer(Rect, val)
|
| DESCRIPTION Return pixel location of a percentage of the rectangle.
|
| PARAMETERS  LPRECT Rect - Specifies pointer to the rectangle.
|        int   val  - Specifies value in persents.
|
| RETURN     The pixel location of a percentage of the rectangle.
|
*/
_LOCAL int StatusBarPer(LPRECT Rect, int val)
{
        return(Rect->left + (int)(((((LONG) val) * ((LONG)(Rect->right - Rect->left))) / 100L));
}


/*-------------------------------------------------------------------------
|
| FUNCTION   _LOCAL void StatusBarDraw(hDC, Rect, Min, Max, Cur, hBrush)
|
| DESCRIPTION Draw the percentage bar for the current value.
|
| PARAMETERS  HDC hDC     - Specifies target DC.
|        LPRECT Rect   - Specifies pointer to the rectangle.
|        int Min      - Specifies
|        int Max      - Specifies
|        int Cur      - Specifies
|        HBRUSH hBrush - Specifies
|
| RETURN     None.
|
*/
_LOCAL void StatusBarDraw(HDC hDC, LPRECT Rect, int Min, int Max, int Cur, HBRUSH
hBrush) {
```

```
        HBRUSH hBrBad;
        HBRUSH hBrGood;
        HANDLE hPrv;
        int Maxp;
        int Minp;

        hBrBad  = CreateSolidBrush(RGB(255, 0, 0)) ;        /* Bad range. */
        hBrGood = CreateSolidBrush(RGB(0, 255, 0)) ;        /* Good range. */

        hPrv = SelectObject(hDC, hBrBad) ;

        Minp = StatusBarPer(Rect, Min);
        if (Min)
        {
                Rectangle(hDC, Rect->left, Rect->top, Minp, Rect->bottom);
        }

        Maxp = StatusBarPer(Rect, Max);
        if (Max != 100)
        {
                Rectangle(hDC, Maxp, Rect->top, Rect->right, Rect->bottom);
        }

        SelectObject(hDC, hBrGood) ;
        Rectangle(hDC, Minp, Rect->top, Maxp, Rect->bottom);

        SelectObject(hDC, hPrv);        /* restore previous selected object. */
        DeleteObject(hBrGood) ;
        DeleteObject(hBrBad ) ;

        /*
        ** Draw the current bar.
        */
        hPrv = SelectObject(hDC, hBrush) ;

        Minp = Rect->top + ((Rect->bottom - Rect->top) / 4);
        Maxp = Rect->top + (((Rect->bottom - Rect->top) * 3) / 4);

        Rectangle(hDC, Rect->left, Minp, StatusBarPer(Rect, Cur), Maxp);
        SelectObject(hDC, hPrv);        /* restore previous selected object. */

}


/*-----------------------------------------------------------------------
|
| FUNCTION    _LOCAL void StatusBars(hDC)
|
| DESCRIPTION Update the data changes to the status window bars.
|
| PARAMETERS  HDC hDC - Specifies target DC.
|
| RETURN      None.
|
*/
_LOCAL void StatusBars(HDC hDC)
```

```c
{
        RECT   rc;
        HBRUSH hBrush;
        HANDLE hFont;
        COLORREF hOldBk;
        char   szWork[PROMPT_LEN + 1];

        if (! (UserGetFlags() & PREF_Confid) && ! (UserGetFlags() & PREF_Volume))
                return;
        if (IsIconic(hwndStatus))
                return;
        /*
        ** Get the new font.
        */
        hFont = SelectObject(hDC, hFontCur) ;


        /*
        ** Get the location of the status bars.
        ** From the client area rectangle get the rectangle for the first bar.
        */

        GetClientRect(hwndStatus, (LPRECT)&rc);

        DrawIcon(hDC, rc.right - GetSystemMetrics(SM_CXICON) - 2, 2, bPause ? hicoMain :
hicoStat);

        rc.left   = PROMPT_LEN * cxStatusText ;
        rc.right -= GetSystemMetrics(SM_CXICON) + 4;
        rc.top    = 2;
        rc.bottom = cyStatusText;

        /*
        ** Always using this brush.
        */
        hBrush = CreateSolidBrush(RGB (0, 0, 255)) ;        /* Current val */
        hOldBk = SetBkColor(hDC, GetSysColor(COLOR_BTNFACE));


        if (UserGetFlags() & PREF_Confid)
        {
                /*
                ** The confidence display bar.
                */
                LoadString(VChInst, IDS_CONFID, (LPSTR)szWork, PROMPT_LEN);
                TextOut(hDC, cxStatusText, rc.top, szWork, lstrlen(szWork));

                StatusBarDraw(hDC, &rc, UserGetConfidence(), 100,
                        vrState.confidence, hBrush);

                /*
                ** Move the rectangle down.
                */
                rc.top    += cyStatusText + 4;
                rc.bottom += cyStatusText + 4;
        }
```

```
if (UserGetFlags() & PREF_Volume)
{
        /*
        ** The volume display bar.
        */
        LoadString(VChInst, IDS_VOLUME, (LPSTR)szWork, PROMPT_LEN);
        TextOut(hDC, cxStatusText, rc.top, szWork, lstrlen(szWork));

        StatusBarDraw(hDC, &rc, iVolumeMin, iVolumeMax,
                vrState.amplitude, hBrush);

        /*
        ** Move the rectangle down.
        */
        rc.top    += cyStatusText + 4;
        rc.bottom += cyStatusText + 4;
}

/*
** Put the old font back.
*/
SelectObject(hDC, hFont);

/*
** Free brush.
*/
DeleteObject(hBrush) ;
SetBkColor(hDC, hOldBk);

}


/*-------------------------------------------------------------------------
|
| FUNCTION   _LOCAL void StatusChange(void)
|
| DESCRIPTION Update status information.
|
| PARAMETERS  None.
|
| RETURN     None.
|
*/
_LOCAL void StatusChange(void)
{

        HDC hDC;
        char   szWork[MAXSTRING + 1];

        if (vrState.confidence >= UserGetConfidence())
        {
                StringLoadParam(szWork, IDS_NEW, (LPSTR)vrState.word[0]);
        }
        else
        {
```

```
                    LoadString(VChInst, IDS_QUERY, szWork, MAXSTRING);
        }
        SetWindowText(hwndStatus, szWork);

        hDC = GetDC(hwndStatus);

        StatusBars(hDC);

        ReleaseDC(hwndStatus, hDC);

}

/*------------------------------------------------------------------
|
| FUNCTION    _LOCAL UINT PhraseFind(szStr)
|
| DESCRIPTION Find phrase in phrase listbox
|
| PARAMETERS  PSTR szStr - Specifies pointer to the phrase.
|
| RETURN      Index in the listbox or LB_ERR.
|
*/
_LOCAL UINT PhraseFind(PSTR szStr)
{
        UINT wIdx;
        LONG lRet;
        char szWord[MAX_SYMBOL_LENGTH];

        wIdx = 0;
        while (1)
        {
                lRet = SendMessage(hwndList, LB_GETTEXT, wIdx, (LONG)(LPSTR)szWord);
                if (lRet == LB_ERR || lRet == NULL)
                        return((UINT)LB_ERR);
                if (! lstrcmpi(szStr, szWord))
                        return(wIdx);
                wIdx ++;
        }
}

/*------------------------------------------------------------------
|
| FUNCTION    _LOCAL UINT CloseCallFind(szStr)
|
| DESCRIPTION Check phrase as a close call number.
|
| PARAMETERS  PSTR szStr - Specifies pointer to the phrase.
|
| RETURN      Index in the listbox or LB_ERR.
|
*/
_LOCAL UINT CloseCallFind(PSTR szStr)
{
        UINT wIdx;
        LONG lRet;
```

```
                UINT wordNum;

                for (wIdx = 0; wIdx < wCloseCallNumber; wIdx ++)
                {
                        wordNum =  wIdx + '1';
                        if (! lstrcmpi(szStr, (char *) &wordNum))
                        {
                                lRet = SendMessage(hwndList, LB_GETITEMDATA, wIdx, NULL);
                                if (lRet == LB_ERR || lRet == NULL)
                                        continue;
                                return(wIdx);
                        }
                }
                return((UINT)LB_ERR);
}

/*---------------------------------------------------------------------
|
| FUNCTION   _LOCAL void PhraseListMove(szStr)
|
| DESCRIPTION Move phrase to the close call list.
|
| PARAMETERS  PSTR szStr - Specifies pointer to the phrase.
|
| RETURN     None.
|
*/
_LOCAL void PhraseListMove(PSTR szStr)
{

        int wIdx;
        WORD wordNum;
        char szWord[MAX_SYMBOL_LENGTH];
        LONG lData;

        if (lstrlen (szStr) == 0)
                return;
        wIdx = PhraseFind(szStr);
        if (wIdx == - 1)
                return;
        SendMessage(hwndList, LB_GETTEXT, wIdx, (LONG)(LPSTR)szWord);
        lData = SendMessage(hwndList, LB_GETITEMDATA, wIdx, NULL);
        SendMessage(hwndList, LB_DELETESTRING, wIdx, NULL);
        SendMessage(hwndList, LB_INSERTSTRING, wCloseCallNumber, (DWORD)(LPSTR)
szWord);
        SendMessage(hwndList, LB_SETITEMDATA, wCloseCallNumber, lData);
        wCloseCallNumber ++;
        wordNum = wCloseCallNumber + '0';

#ifdef DEBUG_DLG
        if (DebugFlag & DEBUG_Recog)
#endif
        SpeechEnable((LPSTR) &wordNum);

}
```

```
/*----------------------------------------------------------------------
|
| FUNCTION    _LOCAL int PhraseListInc(void)
|
| DESCRIPTION Return close call list increment
|
| PARAMETERS  None.
|
| RETURN      Close call list increment
|
*/
_LOCAL int PhraseListInc(void)
{
        RECT Rect;
        int ListSize;
        int CloseCallSize;

        CloseCallSize = wCloseCallNumber * (cyStatusText + 1);
        GetClientRect(hwndStatus, (LPRECT) &Rect);
        ListSize = Rect.bottom - iStatusSizeMin;
        if (ListSize < 0)
        {
                ListSize = 0;
        }
        return((ListSize >= CloseCallSize) ? 0 : CloseCallSize - ListSize);

}


/*----------------------------------------------------------------------
|
| FUNCTION    BOOL PhraseListAdd(szStr, ContextEntry)
|
| DESCRIPTION Add phrase to the phrase list.
|
| PARAMETERS  PSTR szStr     - Specifies pointer to the phrase.
|        int ContextEntry - Specifies index in the context list.
|
| RETURN      TRUE if success.
|
*/
BOOL PhraseListAdd(char * szStr, int ContextEntry)
{
        UINT  wIdx;
        BOOL  bWord = FALSE;

        if (szStr == NULL)
        {
                return(TRUE);
        }

        if (ContextEntry == -1)
        {
                /*
                ** Has no context link so look for one.
                */
                if (PhraseFind(szStr) != -1) return(TRUE);
```

```
                    }

#ifdef DEBUG_DLG
            if (DebugFlag & DEBUG_Recog)
#endif
                    bWord = SpeechEnable(szStr);

            /*
            ** Now add it to the list.
            */
            wIdx = (UINT) SendMessage(hwndList, LB_ADDSTRING, 0, (DWORD)(LPSTR) szStr);
            if (wIdx == (UINT)LB_ERR)
            {
                    return(FALSE);
            }
            SendMessage(hwndList, LB_SETITEMDATA, wIdx, MAKELONG(ContextEntry,
bWord));
            return(TRUE);
}

/*-----------------------------------------------------------------------
|
| FUNCTION    _LOCAL void PhraseListSetup(void)
|
| DESCRIPTION Get the current set of words and give them to the recognizer.
|
| PARAMETERS  None.
|
| RETURN      None.
|
*/
_LOCAL void PhraseListSetup(void)
{
            UINT wIdx;
            RECT rc;

            if (! hwndList)
                    return;

            SendMessage(hwndList, WM_SETREDRAW,   FALSE, 0);
            SendMessage(hwndList, LB_RESETCONTENT, 0, 0);

#ifdef DEBUG_DLG
            if (DebugFlag & DEBUG_Recog)
#endif
                    SpeechDisableAll();                 /* Disable all words. */

            ContextListAdd();                           /* Get context first. */

            if (wCloseCallInc)
            {

                    /*
                    ** Resize window to normal
                    */
                    GetWindowRect(hwndStatus, &rc);
```

```
                    rc.bottom -= wCloseCallInc;
                    wCloseCallInc = 0;
                    MoveWindow(
                            hwndStatus,
                            rc.left,
                            rc.top,
                            rc.right - rc.left,
                            rc.bottom - rc.top,
                            TRUE);
            }

        if (bCloseCallWas)
        {

                /*
                ** Include CloseCall information
                */
                wCloseCallNumber = 0;
                for (wIdx = 0; wIdx < vrState.nWords; PhraseListMove(vrState.word[wIdx ++]));

                if (! IsIconic(hwndStatus))
                {
                        /*
                        ** Should we resize PhraseList ?
                        */
                        wCloseCallInc = PhraseListInc();
                        if (wCloseCallInc) {
                                GetWindowRect(hwndStatus, &rc);
                                MoveWindow(
                                        hwndStatus,
                                        rc.left,
                                        rc.top,
                                        rc.right - rc.left,
                                        rc.bottom - rc.top + wCloseCallInc,
                                        TRUE);
                        }
                }

        }
        SendMessage(hwndList, WM_SETREDRAW,  TRUE, 0);

}

/*------------------------------------------------------------------
|
| FUNCTION    _LOCAL void PhraseDrawItem(LPDRAWITEMSTRUCT lpd)
|
| DESCRIPTION Draw item routine for the status item.
|
| PARAMETERS  LPDRAWITEMSTRUCT lpd - Specifies pointer to the DRAWITEMSTUCT.
|
| RETURN    None.
|
*/
_LOCAL void PhraseDrawItem(LPDRAWITEMSTRUCT lpd)
{
```

```c
HBRUSH hBrush;
int   iBkColor;
int   iTxColor;
char  szWord[2 * MAX_SYMBOL_LENGTH + 50];

if (lpd->itemID == -1)
        return;

if ((lpd->itemState & ODS_SELECTED) && (lpd->itemState & ODS_FOCUS))
{
        iBkColor = COLOR_HIGHLIGHT;
        iTxColor = COLOR_HIGHLIGHTTEXT;
}
else
{
        iBkColor = COLOR_WINDOW;
        iTxColor = COLOR_WINDOWTEXT;
}

SetTextColor(lpd->hDC, GetSysColor(iTxColor));
SetBkColor( lpd->hDC, GetSysColor(iBkColor));

hBrush = CreateSolidBrush(GetSysColor(iBkColor));
FillRect(lpd->hDC, (LPRECT)&(lpd->rcItem), hBrush);
DeleteObject(hBrush);

/*
** Now draw the text.
*/
SendMessage(hwndList, LB_GETTEXT, lpd->itemID, (LONG)(LPSTR)szWord);
if (bCloseCallWas && lpd->itemID < wCloseCallNumber)
{
        PaintBitmap(
                lpd->hDC, lpd->rcItem.left, lpd->rcItem.top,
                BMP_SIZE, BMP_SIZE,
                hbmpAnd, hbmpPaint, lpd->itemID * BMP_SIZE, 0);
        TextOut(lpd->hDC, lpd->rcItem.left + BMP_SIZE, lpd->rcItem.top, szWord,
lstrlen(szWord));
}
else
{
        if (!HIWORD(SendMessage(hwndList, LB_GETITEMDATA, lpd->itemID, 0L)))
        {
                SetTextColor(lpd->hDC, GetSysColor(COLOR_GRAYTEXT));
        }

#ifdef DEBUG_DLG
        if (DebugFlag & DEBUG_ContFull)
        {
                TabbedTextOut(lpd->hDC, lpd->rcItem.left, lpd->rcItem.top,
                        szWord, lstrlen(szWord), 2, ContextTabs, lpd->rcItem.left);
        }
        else
#endif

        TextOut(lpd->hDC, lpd->rcItem.left, lpd->rcItem.top, szWord, lstrlen(szWord));
```

```
        }

}

/*-------------------------------------------------------------------
 |
 | FUNCTION   _LOCAL void PhraseExec(wIdx)
 |
 | DESCRIPTION Execute the links associated with the phrase.
 |
 | PARAMETERS  UINT wIdx - Specifies index of the phrase in the listbox.
 |
 | RETURN     None.
 |
 */
_LOCAL void PhraseExec(UINT wIdx)
{
        if (wIdx != (UINT)LB_ERR)
        {

                StatusChange();
                /*
                ** Activate the context link macro. If it has one.
                */
                ContextListSelect(LOWORD(SendMessage(hwndList, LB_GETITEMDATA,
wIdx, NULL)));
        }

}

/*-------------------------------------------------------------------
 |
 | FUNCTION   void CALLBACK PhraseTimerProc(hwnd, msg, idTimer, dwTime)
 |
 | DESCRIPTION An application-defined callback function that
 |        processes WM_TIMER messages.
 |        Look for context change
 |
 | PARAMETERS  HWND  hwnd          - Identifies the window associated with the timer.
 |        UINT  msg      - Specifies the WM_TIMER message.
 |        UINT  idTimer - Specifies the timer's identifier.
 |        DWORD dwTime         - Specifies the current system time.
 |
 | RETURN     None.
 |
 */
void CALLBACK  PhraseTimerProc(HWND hwnd, UINT wMsg, UINT idTimer, DWORD dwTime)
{
        static BOOL Active = FALSE;

        if (!Active)
        {
                Active = TRUE;
                if (ContextCheck(FALSE))
                {
                        bCloseCallWas = FALSE;
```

```
                        SpeechErase();
                        PhraseListSetup();          /* rebuild the current vocab list. */
                }
                        Active = FALSE;
                }
        }

        /*---------------------------------------------------------------
        |
        | FUNCTION   _LOCAL void StartTimer(void)
        |
        | DESCRIPTION Start timer to look to the context change.
        |
        | PARAMETERS  None.
        |
        | RETURN     None.
        |
        */
        _LOCAL void StartTimer(void)
        {
                PhraseTimer = SetTimer(NULL, IDT_PHRASE, 500, (TIMERPROC)PhraseTimerProc);
                if (! PhraseTimer)
                        Error(ERRNoTimers);
        }

        /*---------------------------------------------------------------
        |
        | FUNCTION   _LOCAL void StopTimer(void)
        |
        | DESCRIPTION Stop(kill) timer.
        |
        | PARAMETERS  None.
        |
        | RETURN     None.
        |
        */
        _LOCAL void StopTimer(void)
        {
                KillTimer(NULL, PhraseTimer);
        }

        /*---------------------------------------------------------------
        |
        | FUNCTION   void StatusSetPref(HWND hwnd)
        |
        | DESCRIPTION Set the windows preferences.
        |             Find the minimum size for the status window.
        |             number of pixel height units to the start of the vocab box.
        |
        | PARAMETERS  HWND hwnd - Specifies handle to the status window.
        |
        | RETURN     None.
        |
        */
        void StatusSetPref(HWND hwnd)
        {
```

```c
        int  sfNew = UserGetFlags();
        RECT rc;
        int  yInc = 0;


        iStatusSizeMin = 0;
        if (sfNew & PREF_Volume)
                iStatusSizeMin += 4 + cyStatusText;
        if (sfNew & PREF_Confid)
                iStatusSizeMin += 4 + cyStatusText;
        iStatusSizeMin = max(iStatusSizeMin, 6 + GetSystemMetrics(SM_CYICON));
        GetClientRect(hwnd, &rc);
        if (rc.bottom < iStatusSizeMin)
                yInc = iStatusSizeMin - rc.bottom;
        GetWindowRect(hwnd, &rc);
        MoveWindow(
                hwnd,
                rc.left,
                rc.top,
                rc.right - rc.left,
                rc.bottom - rc.top + yInc,
                TRUE);
        SendMessage(hwnd, WM_SIZE, 0, 0L);
        InvalidateRect(hwnd, NULL, TRUE) ; /* rebuild if resized or not */
        ContextCheck(TRUE);
        PhraseListSetup();              /* rebuild the current vocab list. */

}

/*---------------------------------------------------------------------
|
| FUNCTION   _LOCAL void SelectOurFont()
|
| DESCRIPTION Select font for phrase listbox.
|
| PARAMETERS None.
|
| RETURN    None.
|
*/
_LOCAL void SelectOurFont()
{

        HDC      hDC;
        TEXTMETRIC tm;
        HFONT hFontNew;

        hFontNew = UserGetFont();

        hDC  = CreateIC((LPSTR)"DISPLAY", NULL, NULL, NULL) ;
        SelectObject(hDC, hFontNew) ;
        GetTextMetrics(hDC, &tm);

        SendMessage(hwndList, WM_SETFONT, hFontNew, 0L);
```

```
            SendMessage(hwndList, LB_SETITEMHEIGHT, 0, MAKELONG(max(tm.tmHeight,
    BMP_SIZE), 0));
            cxStatusText = tm.tmAveCharWidth;
            cyStatusText = tm.tmHeight;
            if (hFontCur != NULL)
                    DeleteObject(hFontCur);
            hFontCur = hFontNew;
            DeleteDC(hDC);

    }


    /*------------------------------------------------------------------------
    |
    | FUNCTION    BOOL CALLBACK StatusWndProc(hwnd, wMsg, wParam, lParam)
    |
    | DESCRIPTION  Window Proc VoiceStatus class.
    |            The form of the status window is follows:
    |              Title bar = System menu icon, last word, w/ current
    |              Confidence
    |              Volume
    |              Current options list box.
    |
    | PARAMETERS   HWND hwnd   - Specifies the handle of the window
    |         UINT wMsg   - Specifies the message
    |         WORD wParam - Specifies 16 bits of additional
    |                 message-dependent information
    |         LONG lParam - Specifies 16 bits of additional
    |                 message-dependent information
    |
    | RETURN     Depend upon the message.
    |
    */
    long FAR PASCAL StatusWndProc(HWND hwnd, UINT wMsg, WORD wParam, LONG lParam)
    {
            static WORD   wMenuCmd = NULL;
            static DWORD  dwMenuBits = NULL;
            static BOOL  bRecogReady = FALSE;

            switch (wMsg)
            {
                    case WM_CREATE:
                    {
                            /* Install System
                            */
                            LPCREATESTRUCT lpcs = (LPCREATESTRUCT) lParam;

                            /* Create the list of available words for the user.
                            */
                            hwndList = CreateWindow(
                                    "LISTBOX",
                                    NULL,
                                    WS_CHILD | WS_VISIBLE | WS_BORDER |
                                    WS_HSCROLL | LBS_NOINTEGRALHEIGHT |
                                    LBS_NOTIFY | LBS_OWNERDRAWFIXED |
                                    LBS_HASSTRINGS | LBS_WANTKEYBOARDINPUT,
                                    iStatusSizeMin,
```

```c
                                        0,
                                        lpcs->cx - iStatusSizeMin,
                                        lpcs->cy,
                                        hwnd,                           /* parent. */
                                        IDLIST_PHRASE,
                                        VChInst,
                                        (LPSTR) NULL);
                        if (hwndList == NULL)
                        {
                                        return(-1);
                        }

                        /* Hook message queue
                        */
                        HookInstall(TRUE);

                        /* Start DDE with Program Manager
                        */
                        ShellDdeInit(&VCTalk);

                        /* Install help hook (F1 in dialogs and menu)
                        */
                        HelpHookInit();

                        /* The window gets created, so do the one time stuff.
                        */
                        hicoMain  = LoadIcon(VChInst, MAKEINTRESOURCE(ICO_MAIN));
                        hicoStat  = LoadIcon(VChInst, MAKEINTRESOURCE(ICO_STAT));
                        hbmpPaint = LoadBitmap(VChInst,
MAKEINTRESOURCE(BMP_CLCALL));
                        hbmpAnd   = CreateAndBitmap(hbmpPaint);

#ifdef DEBUG_DLG
                        /* Update system menu
                        */
                        {
                                        char szWork[MAXSTRING + 1];
                                        HMENU hMenu = GetSystemMenu(hwnd, FALSE);;

                                        AppendMenu(hMenu, MF_SEPARATOR, 0, 0);
                                        LoadString(VChInst, IDS_DEBUG, (LPSTR)szWork,
MAXSTRING);

                                        AppendMenu(hMenu, MF_STRING, IDM_SYSDEBUG,
(LPSTR)szWork);

                                        DrawMenuBar(hwnd);
                        }
#endif

                        /* Set prefs
                        */
                        SelectOurFont();
                        StatusSetPref(hwnd);

                        /* Status is owner of the speech channal
                        */
                        SpeechOwner(hwnd);
```

```
                          /* Set the initial values to the prase list.
                          */
                          PhraseListSetup():

                          bRecogReady = TRUE;

                          /* Do not put break here.
                          ** We change user from void to current
                          */
                  }

                  case VCM_USERCHANGED:
                  {
                          RECT rc;
                          HCURSOR hcur;
                          HWND hwndEdit;

                          hcur = SetCursor(LoadCursor(NULL, IDC_WAIT));

                          /* Set Status placement
                          */
                          UserGetWinRect(szStatusClass, &rc);
                          MoveWindow(
                                  hwnd,
                                  rc.left,
                                  rc.top,
                                  rc.right - rc.left,
                                  rc.bottom - rc.top,
                                  TRUE);
                          StopTimer();
                          bRecogReady = FALSE;

                          /* Load voice file
                          */
#ifdef DEBUG_DLG
                          if (DebugFlag & DEBUG_Recog)
#endif
                          SpeechUserChange();

                          /* Load Language
                          */
                          hwndEdit = FindWindow(szFrameClass, NULL);
                          if (hwndEdit != NULL)
                          {
                                  /* Load from the editor
                                  */
                                  ContextNewLang((LPLANG)SendMessage(hwndEdit,
iEditChangeMsg, 0, 0L));
                          }
                          else
                          {
                                  /* Load from the file
                                  */
                                  ContextNewLang(NULL);
                          }
```

```c
                bRecogReady = TRUE;
                StartTimer();
                SetCursor(hcur);
                PhraseListSetup();
                break;
        }

        case WM_MENUSELECT:
                /* Keep menu selection for help
                */
                dwMenuBits=lParam;
                wMenuCmd=wParam;
                goto defmsg;

        case VCM_HELP:
                if (!(LOWORD(dwMenuBits) & MF_POPUP))
                {
                        if (!(LOWORD(dwMenuBits) & MF_SYSMENU))
                        {
                                /* Menu help
                                */
                                Help(hwnd, HELP_VCMenuPrefs + wMenuCmd -
MENU_STATUS);
                        }
                        else
                        {
                                /* System menu help
                                */
                                Help(hwnd, HELP_SysMenu);
                        }
                }
                else
                {
                        /* General help
                        */
                        Help(hwnd, HELP_Status);
                }
                break;

        case VCM_SPEECH:
        {
                /*
                ** Speech available
                */
                UINT wIdx;
                UINT wUtt;

                if (bRecogReady && !bPause)
                {
                        bRecogReady = FALSE;
                        StopTimer();
                        wUtt = SpeechRecog(&vrState);

                        /* Check Close Call list first.
```

```c
                                                    */
                                                    if (wUtt != 0)
                                                    {
                                                            if (vrState.confidence >= UserGetConfidence()) {
                                                                    if (bCloseCallWas) {
                                                                            wIdx = CloseCallFind(vrState.word[0]);
                                                                            if (wIdx != (UINT)LB_ERR) {
                                                                                    SendMessage(hwndList,
LB_GETTEXT, wIdx, (LONG)(LPSTR)(vrState.word[0]));

                                                                                    if ( UserGetFlags() &
PREF_Adapt) {

                                                                                            SpeechAdapt(vrState.w
ord[0], wUttCloseCall);

                                                                                    }
                                                                            }
                                                                            else {
                                                                                    wIdx =
PhraseFind(vrState.word[0]);

                                                                            }
                                                                    }
                                                                    else {
                                                                            wIdx = PhraseFind(vrState.word[0]);
                                                                    }
                                                                    bCloseCallWas = FALSE;
                                                                    SpeechErase();

                                                                    /* A word was recognized correctly.
                                                                    */
                                                                    PhraseExec(wIdx);
                                                            }
                                                            else {
                                                                    /* Setup Close Call list
                                                                    */
                                                                    bCloseCallWas = TRUE;
                                                                    wUttCloseCall = wUtt;
                                                                    StatusChange();
                                                                    PhraseListSetup();
                                                            }
                                                    }
                                                    StartTimer();
                                                    bRecogReady = TRUE;
                                            }
                                            break;
                            }

                            case VCM_TRAIN:
                            {
                                    /* Word was trained
                                    */
                                    UINT wIdx;
                                    LONG lData;
                                    RECT rc;

                                    wIdx = PhraseFind((PSTR)lParam);
                                    if (wIdx != (UINT)LB_ERR) {
                                            lData = SendMessage(hwndList, LB_GETITEMDATA, wIdx, 0L);
```

```c
                        if (!HIWORD(lData))  {
                                SendMessage(hwndList, LB_SETITEMDATA, wIdx,
MAKELONG(LOWORD(lData), TRUE));
                                SendMessage(hwndList, LB_GETITEMRECT, wIdx,
(LONG)(LPRECT)&rc);
                                InvalidateRect(hwndList, &rc, TRUE);
                        }
                }
                break;
        }

        case WM_PAINT:
        {
                /* A repaint instruction has been given.
                */
                HDC        hDC;
                PAINTSTRUCT ps;
                HICON      hIcon;

                hDC = BeginPaint(hwndStatus, (LPPAINTSTRUCT)&ps);
                if (IsIconic(hwndStatus))
                {
                        /* Draw iconic window
                        */
                        hIcon = (bPause) ? hicoMain : hicoStat;
                        DrawIcon(hDC, 0, 0, hIcon);
                }
                else
                {
                        /* Create the volume and confidence boxes.
                        */
                        StatusBars(hDC);
                }
                EndPaint(hwndStatus, (LPPAINTSTRUCT)&ps);
                break;
        }

        case WM_SIZE:
        {
                /* Move the phrase list.
                */
                RECT rc;

                GetClientRect(hwnd, &rc);
                MoveWindow(
                        hwndList,
                        rc.left,
                        rc.top + iStatusSizeMin,
                        rc.right - rc.left + 1,
                        rc.bottom - rc.top - iStatusSizeMin + 1,
                        TRUE);
                break;
        }

        case WM_GETMINMAXINFO:
        {
```

```c
                MINMAXINFO FAR * lpmmi = (MINMAXINFO FAR *) lParam;
                RECT rc;

                memset(&rc, 0, sizeof(rc));
                rc.bottom = iStatusSizeMin;
                AdjustWindowRect(&rc, WS_OVERLAPPEDWINDOW, TRUE);

                lpmmi->ptMinTrackSize.x = MAX_SYMBOL_LENGTH * cxStatusText;
                lpmmi->ptMinTrackSize.y = rc.bottom - rc.top + wCloseCallInc;
                break;
        }

        case WM_SETFOCUS:
                /* We just got the focus.
                */
                SetFocus(hwndList);              /* Give it to the list box. */
                break;

        case WM_QUERYDRAGICON:
                /* A repaint instruction has been given.
                */
                return(bPause ? hicoMain : hicoStat);

        case WM_DRAWITEM:
                /* The system listbox wants us to draw the item.
                ** DRAWITEMSTRUCT
                */
                PhraseDrawItem((LPDRAWITEMSTRUCT) lParam);
                break;

#ifdef DEBUG_DLG
        case WM_SYSCOMMAND:
                if ((wParam & 0xFFF0) == IDM_SYSDEBUG)
                {
                        /* Bring up the Debug dialog box.
                        */
                        DialogBox(VChInst, MAKEINTRESOURCE(DLG_DEBUG),
hwnd, DebugDlgProc);

                        /* Rebuild phrase list
                        */
                        PhraseListSetup();
                }
                else
                {
                        goto defmsg;

                }
                break;
#endif

        case WM_COMMAND:
                switch (wParam)
                {
                        case IDM_PREFS:
                                /* Bring up the User Prefereces dialog box.
```

```
                                        */
                                        if(UserPref(hwnd))
                                        {
                                                SelectOurFont();
                                        }
                                        StatusSetPref(hwnd);
                                        break;

                        case IDM_TRAIN:
                                        /* Bring up the Vocabulary Training dialog box.
                                        */
                                        SendMessage(hwnd, WM_COMMAND,
IDLIST_PHRASE, MAKELONG(0, LBN_DBLCLK));
                                        break;

                        case IDM_PAUSE:
                        {
                                        /* Pause on/off.
                                        */
                                        char szTitle[MAXSTRING + 1];

                                        bPause = ! bPause;
                                        CheckMenuItem(GetMenu(hwnd), IDM_PAUSE,
                                                MF_BYCOMMAND | (bPause ? MF_CHECKED
: MF_UNCHECKED));

                                        LoadString(VChInst, IDS_TITLE, (LPSTR)szTitle,
sizeof(szTitle));

                                        SetWindowText(hwnd, (LPSTR)szTitle);
                                        InvalidateRect(hwnd, NULL, TRUE) ;
                                        break;
                        }

                        case IDM_EDIT:
                        {
                        /*
                                        ** Bring up the Language Editor
                                        */
                                        char szVeFile[MAXFILENAME + 1];

                                        IniGetVeFile(szVeFile);
                                        WinExec(szVeFile, SW_SHOW);
                                        break;

                        }

                        case IDM_EXIT:
                                        /* Exit now
                                        */
                                        SendMessage(hwnd, WM_CLOSE, 0, 0L);
                                        break;

                        case IDM_HELPCONTENT:
                                        /* Bring up the Help
                                        */
                                        Help(hwnd, HELP_Status);
                                        break;
```

```
                        case IDM_HELPSEARCH:
                            /* Bring up the Help Search
                            */
                            Help(hwnd, HELP_Search);
                            break;

                        case IDM_HELPONHELP:
                            /* Bring up the HelpOnHelp
                            */
                            Help(hwnd, HELP_OnHelp);
                            break;

                        case IDM_ABOUT:
                            /* Bring up the About.. dialog box.
                            */
                        About(hwnd);
                        break;

                        case IDLIST_PHRASE:
                            switch (HIWORD(IParam)) {
                                case LBN_DBLCLK:
#ifdef DEBUG_DLG
                                        if (DebugFlag & DEBUG_Force) {
                                                /* Execute command
                                                */
                                                char * Ptr;
                                                UINT wIdx =
(UINT)SendMessage(hwndList, LB_GETCURSEL, 0, 0);

                                                vrState.confidence = 100;
                                                vrState.amplitude  = 0;
                                                SendMessage(hwndList,
LB_GETTEXT, wIdx, (LONG)(LPSTR)(vrState.word[0]));

                                                if (DebugFlag &
DEBUG_ContFull)
                                                {
                                                        /* Skip debug
information
                                                        */
                                                        for (Ptr =
vrState.word[0]; *Ptr; Ptr ++)
                                                        {
                                                                if (* Ptr == '\t')
                                                                {
                                                                        * Ptr =
'\0';
                                                                        break;
                                                                }
                                                        }
                                                }
                                                PhraseExec(wIdx);
                                                break;
                                        }
#endif
                                        /* Train command
                                        */
```

```c
                                                       TrainExec(TRUE,
(UINT)SendMessage(hwndList, LB_GETCURSEL, 0, 0), hwndList);
                                                       break;
                                       case LBN_SETFOCUS:
                                               /* We just got focus. clear previous
inputs.
                                               */
                                               break;
                               default :
                                       goto defmsg;
                       }
                       break;

               default:
                       goto defmsg;

               }
               break;

       case WM_QUERYENDSESSION:
       {
               WINDOWPLACEMENT wndpl;
               HWND hwndEdit;

               if (wParam == 2)
               {
                       /* We don't quit, just hange user
                       */
                       hwndEdit = FindWindow(szFrameClass, NULL);
                       if (hwndEdit != NULL)
                       {
                               Error(ERREditExist);
                               ShowWindow(hwndEdit, SW_SHOWNORMAL);
                               SetFocus(hwndEdit);
                               break;
                       }

               }
               /* Save users settings
               */
               wndpl.length = sizeof(wndpl);
               GetWindowPlacement(hwnd, &wndpl);
               UserSetWinRect(szStatusClass, &(wndpl.rcNormalPosition));
               goto defmsg;
       }

       case WM_CLOSE:
               /* Ask permision before quit
               */
               if (CallTaskWindows(TRUE, WM_QUERYENDSESSION, TRUE, 0L))
               {
                       CallTaskWindows(FALSE, WM_DESTROY, 0, 0L);
               }
               break;

       case WM_DESTROY :
```

```
                {
                        /* Free resores
                        */
                        HCURSOR hcur;

                        hcur = SetCursor(LoadCursor(NULL, IDC_WAIT));
                        StopTimer();
                        DestroyIcon(hicoStat);
                        DestroyIcon(hicoMain);
                        DeleteObject(hbmpPaint);
                        DeleteObject(hbmpAnd);
                        DeleteObject(hFontCur);

                        /* Free speech system
                        */
#ifdef DEBUG_DLG
                        if (DebugFlag & DEBUG_Recog)
#endif
                        SpeechFree();

                        /* Unhook message queue
                        */
                        HookInstall(FALSE);
                        HookFreeJournal();

                        /* Close help if was opened
                        */
                        Help(hwnd, HELP_Quit);

                        /* Stop DDE
                        */
                        ShellDdeExit(&VCTalk);

                        /* Unhook help hook
                        */
                        HelpHookExit();

                        /* Save the user file.
                        */
                        UserExit();
                        SetCursor(hcur);

                        /* Kill the task and other windows.
                        */
                        PostQuitMessage(0);
                        break;

                }
                default:
                        if (wMsg == iEditChangeMsg)
                        {
                                /* Changes in Editor saved
                                ** We need to update language
                                */
                                HCURSOR hcur;
```

```c
                                hcur = SetCursor(LoadCursor(NULL, IDC_WAIT));
                                StopTimer();
                                bRecogReady = FALSE;

                                /* Load Language
                                */
                                ContextNewLang((LPLANG)lParam);
                                bRecogReady = TRUE;
                                StartTimer();
                                SetCursor(hcur);
                                break;
                        }

                defmsg:
                        return DefWindowProc(hwnd, wMsg, wParam, lParam);
        }

        return (NULL);

}

/*-------------------------------------------------------------------
|
| FUNCTION    BOOL StatusInit(BOOL bNew)
|
| DESCRIPTION
|
| PARAMETERS
|
| RETURN
|
*/
BOOL StatusInit(BOOL bNew)
{

        WNDCLASS wc;
        char     szTitle[MAXSTRING + 1];
        RECT     rc;
        HWND     hwnd;

        if (bNew)
        {
                UserInit();

                /* To reload file
                */
                iEditChangeMsg = RegisterWindowMessage(szFrameClass);

                /* Register the window class.
                */
                memset(&wc, 0, sizeof(wc));              /* zero structure to start. */

                wc.style       = CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW ;
                wc.lpfnWndProc = (WNDPROC)StatusWndProc;
                wc.hInstance   = VChInst;                        /* task owner. */
                wc.hCursor     = LoadCursor(NULL, IDC_ARROW) ;
```

```c
        wc.hbrBackground = COLOR_BTNFACE + 1;
        wc.lpszClassName = (LPSTR) szStatusClass;
        wc.lpszMenuName  = MAKEINTRESOURCE(MENU_STATUS);

        if (! RegisterClass(&wc))
                return(FALSE);

        hAccTableStatus = LoadAccelerators(VChInst,
MAKEINTRESOURCE(ATBL_STATUS));


        /* Create Status Window
        */
        UserGetWinRect(szStatusClass, &rc);
        LoadString(VChInst, IDS_TITLE, (LPSTR)szTitle, sizeof(szTitle) - 1);

        hwndStatus = CreateWindowEx(
                WS_EX_TOPMOST,
                szStatusClass,
                (LPSTR)szTitle,
                WS_OVERLAPPEDWINDOW & (~WS_MAXIMIZEBOX),
                rc.left,
                rc.top,
                rc.right  - rc.left,
                rc.bottom - rc.top,
                NULL,
                NULL,
                VChInst,
                (LPSTR) NULL);

        if (! hwndStatus)
                return(FALSE);

        /* Send timer message to update context.
        ** every 1/2 of a second or so.
        */
        StartTimer();
        ShowWindow(hwndStatus, SW_SHOWNORMAL);

/* Install recognition system
*/
#ifdef DEBUG_DLG
        if (DebugFlag & DEBUG_Recog)
#endif

                SpeechInit();

                PhraseListSetup();
        }
        else
        {
        /* Only one instanse of Voice Control should be present
        */
        hwnd = FindWindow(szStatusClass, NULL);
        if (hwnd)
        {
                /* This should always be true !?
```

```c
                                  */
                    ShowWindow(hwnd, SW_SHOWNORMAL);

                    /* Flash it to indicate location.
                    */
                    SetFocus(hwnd);
                }

            }

        return(TRUE);

    }

/*------------------------------------------------------------------
|
| FUNCTION    BOOL StatusCheckMsg(MSG * pMsg)
|
| DESCRIPTION Message translation.
|
| PARAMETERS  MSG * pMsg - Specifies pointer to the incoming message.
|
| RETURN      TRUE if processed(message belong to the status).
|
*/
BOOL StatusCheckMsg(MSG * pMsg)
{

        if (hwndStatus != NULL && GetFocus() == hwndList &&
                TranslateAccelerator(hwndStatus, hAccTableStatus, pMsg))
                return(TRUE);

        return(FALSE);
}

/*------------------------------------------------------------------
|
| FUNCTION    HWND StatusGetWindow(void)
|
| DESCRIPTION Return status window handle.
|
| PARAMETERS  None.
|
| RETURN      Window handle.
|
*/
HWND StatusGetWindow(void)
{
        return(hwndStatus);
}
```